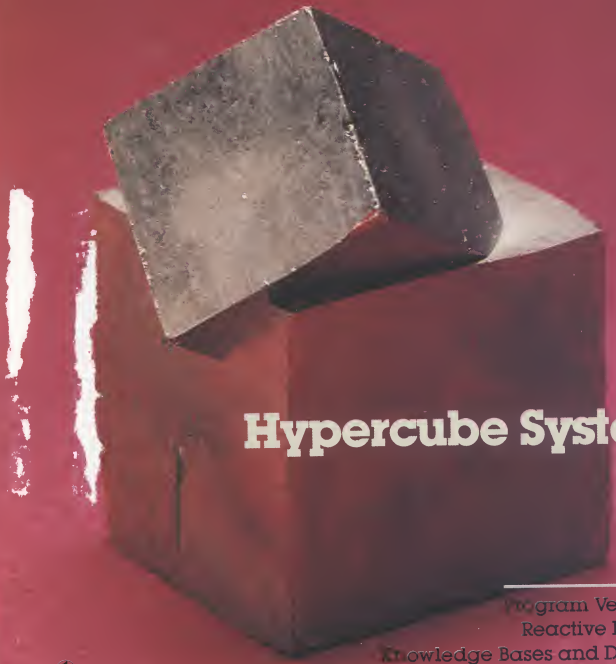
NOVEMBER 1990

COMPUTER



Hypercube Systems

Plus:

Program Verification
Reactive Keyboard
Knowledge Bases and Databases
Application Portability
Interactive User Interfaces

IEEE COMPUTER SOCIETY

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

Preview the product that will change Ada forever

$149 INTRODUCTORY PRICE

AdaZ® opens the Ada world to PC programmers, setting off a tide of sweeping change that will rock Ada the way another innovator shook up Pascal. With a menu-driven interface, a complete set of tools, ACVC validation, and an incredibly low price, AdaZ breaks down barriers that have kept some programmers from exploring the fastest growing language in the industry. You can explore it now for $149.

Professional From A to Z—Despite the unprecedented low price, AdaZ is a serious compiler. Created by Meridian Software Systems, AdaZ comes from a long line of acclaimed professional development systems. It's based directly on Meridian Ada for the PC, a system used successfully for countless defense and commercial market projects.

In fact, *PC Week* ranks AdaZ technology among top developmental systems. Finishing just a fraction of a point behind Alsys, Inc.'s 386 DOS Ada Development System, AdaZ sells for one tenth its price. One tenth.

Gets You From A to Z Fast—AdaZ answers the industry's need for a quick-start Ada compiler. With hyper-text on-line help and a windowed editor, you can explore Ada within the first few hours on this innovative system.

Includes Everything From A to Z—Derived from customer suggestions, AdaZ is an affordable Ada desktop development system without limitations. It includes:

■ compiler/linker ■ debugger ■ optimizer ■ utility and DOS environment library ■ Amake ■ graphic interface ■ windowed editor ■ on-line LRM

You can't afford to pass up a chance to experience AdaZ. Act now, because in 60 days, you'll have to pay $495 for this exciting product.

Meridian's Open Ada policy makes your AdaZ purchase an investment in your future. You can trade-up to a full line of Ada development products. And you receive full value for your AdaZ purchase for two years. Customer support is also available. But you must order now. **Call 1(800)221-2522.**

Meridian Software Systems, Inc. 10 Pasteur St., Irvine, CA 92718

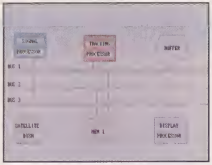
AdaZ

Alsys Inc. is 386 DOS Ada Development System is a trademark of Alsys Inc. AdaZ is a registered trademark of Meridian Software Systems, Inc.

Six Simulation Solutions

Now you can predict system performance **before** costly implementation

NETWORK II.5

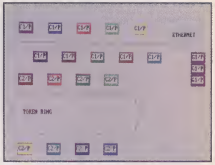


Predict computer/communications network performance. You graphically define your network of processors and storage devices, interconnected by communication channels. Animation follows immediately—no programming.

COMNET II.5

Predict LAN, LAN/WAN, voice, and data network performance. You graphically define the topology of your network through routing nodes and transmission links. Animation follows immediately—no programming.

LANNET II.5



Predict LAN performance. You graphically define your network of stations and gateways interconnected by LAN's. Animation follows immediately—no programming.

SIMSCRIPT II.5

A widely used programming language for development of process-oriented models. This English-like language, with built-in graphics, is in use at 4,126 sites worldwide.

MODSIM II

A new object-oriented programming language with built-in graphics. It incorporates the key elements of modern software engineering: block structure, modularity, and strong typing.

SIMFACTORY II.5

Predict factory performance. You graphically describe your factory through an easy-to-use interface. Animation follows immediately—no programming.

Free Trial and Training Offer

For over 28 years CACI has provided its simulation software on a free trial basis—**no cost, no obligation**. You get everything you need to try the product of your choice on your PC, Workstation, or Mainframe.

For Immediate Information:

In the U.S., call (619) 457-9681 or Fax (619) 457-1184. In Europe, call Nigel McNamara, in our UK office, on (081) 332-0122 or Fax (081) 332-0112. Worldwide, call Joe Lenz, in the U.S.A., at (619) 457-9681 or Fax (619) 457-4463.

Rush free trial and training information on:

- ☐ **SIMSCRIPT II.5®**
- ☐ **MODSIM II®**
- ☐ **SIMFACTORY II.5®**
- ☐ **NETWORK II.5®**
- ☐ **COMNET II.5®**
- ☐ **LANNET II.5™**

☐ Send me information on your special university program for faculty only.

Name _____

Company _____

Address _____

_____

_____

Telephone _____ Fax _____

Computer _____ Op. Syst. _____

Mail or Fax this coupon to:

CACI Products Company
3344 North Torrey Pines Court
La Jolla, California 92037
Phone (619) 457-9681
Fax (619) 457-1184

In Europe:
CACI Products Division
Palm Court, 4 Heron Square
Richmond, Surrey, TW9 1EW
United Kingdom
Phone (081) 332-0122
Fax (081) 332-0112

SIMSCRIPT II.5, SIMFACTORY II.5, COMNET II.5, MODSIM II, and NETWORK II.5 are registered trademarks and service marks; LANNET II.5 is a trademark of CACI Products Company. ©1990 CACI Products Company.

BEE COMP

COMPUTER

November 1990 Published by the IEEE Computer Society Vol. 23, No. 11

ARTICLES

13 Multiprocessor Algorithms for Relational-Database Operators on Hypercube Systems

Ophir Frieder

As databases expand and applications become more diverse, demands on computational engines supporting database processing increase. With appropriate algorithms, commercially available systems can meet the demands.

30 A Functional Correctness Model of Program Verification

Marvin V. Zelkowitz

This model's verification conditions depend only on elementary symbolic execution of a trace table. An easy-to-learn technique, it's used in a freshman computer science course.

41 The Reactive Keyboard: A Predictive Typing Aid

John J. Darragh, Ian H. Witten, and Mark L. James

A new interface that partially automates typed input increases the speed and ease of communicating with a computer, especially for physically disabled people.

51 Knowledge Bases and Databases: Converging Technologies, Diverging Interests

Yehudah Freundlich

Knowledge-based technology brought the perspective of knowledge to applications programming and database modeling. But KB specialists focus on the knowledge level, whereas database specialists focus on the data level.

59 Strategies for Supporting Application Portability

James D. Mooney

Many strategies are available to improve application portability. System designers and application designers must work together to produce the best solutions.

71 Garnet: Comprehensive Support for Graphical, Highly Interactive User Interfaces

Brad A. Myers, Dario A. Giuse, Roger B. Dannenberg, Brad Vander Zanden, David S. Kosbie, Edward Pervin, Andrew Mickish, and Philippe Marchal

Garnet helps create highly interactive user interfaces by emphasizing easy specification of object behavior, often by demonstration and without programming.

Circulation: *Computer* (ISSN 0018-9162) is published monthly by the IEEE Computer Society, 10662 Los Vaqueros Circle, PO Box 3014, Los Alamitos, CA 90720-1264; phone (714) 821-8380. IEEE Computer Society Headquarters, 1730 Massachusetts Ave. NW, Washington, DC 20036-1903; IEEE Headquarters, 345 East 47th St., New York, NY 10017. Annual subscription included in society member dues. Nonmember subscription rates: available upon request. Single copy prices: members $10.00; nonmembers $20.00. This magazine is also available in microfiche form.

Postmaster: Send undelivered copies and address changes to *Computer*, IEEE Service Center, 445 Hoes Lane, Piscataway, NJ 08855. Second class postage is paid at New York, New York, and at additional mailing offices.

Copyright and reprint permission: Copyright © 1990 by the Institute of Electrical and Electronics Engineers, Inc. All rights reserved. Abstracting is permitted with credit to the source. Libraries are permitted to photocopy beyond the limits of US copyright law for private use of patrons: (1) those post-1977 articles that carry a code at the bottom of the first page, provided the per-copy fee indicated in the code is paid through the Copyright Clearance Center, 29 Congress Street, Salem, MA 01970; (2) pre-1978 articles without fee. Instructors are permitted to photocopy isolated articles for noncommercial classroom use without fee. For other copying, reprint, or republication permission, write to Permissions Editor, *Computer*, 10662 Los Vaqueros Circle, PO Box 3014, Los Alamitos, CA 90720-1264.

Editorial: Unless otherwise stated, bylined articles, as well as product and service descriptions, reflect the author's or firm's opinion. Inclusion in *Computer* does not necessarily constitute endorsement by the IEEE or the Computer Society. All submissions are subject to editing for style, clarity, and space considerations.

BPA

DEPARTMENTS

6 President's Message
Why we should care about standards

10 Letters to the Editor
Mea culpa

87 Standards
An American's view of the Japanese standards system

91 Computer Society News
News from the Committee on Public Policy: Identifying and mitigating risks to computer resources; Service awards presented at ICCD and ITC; Gallager receives IEEE Medal of Honor

93 Update

94 Product Reviews
Object-oriented programming: Smalltalk-80 and Objectworks; Hardware to support graphical user interfaces; Microsoft C 6.0; A combined mouse, trackball, keypad

102 New Products

107 IC/Microsystem Announcements

109 Conferences/Call for Papers/Calendar

131 Book Reviews

135 CS Magazines

Cover photo: Unlike the pyrite crystals on the cover, hypercube systems linked to relational databases could actually prove worth their weight in gold. Janice Sheldon/Photo 20-20
Cover design: Jay Simpson, Design & Direction

In the next issue

Philosophies for Engineering Computer-Based Systems; Are Parallel Languages Responsive to the Needs of Scientific Programmers?; Architecture-Independent Parallel Computation; An Overview of Common Benchmarks; Compiling Scientific Code Using Partial Evaluation; Paradigm: A Highly Scalable, Shared-Memory Multiprocessor Architecture

Career Opportunities, 120; Computer Society Information, 136; Membership Application, 130; Change-of-Address Form, p. 101; Advertiser/Product Index, 104; Reader Service Card, 104A

COMPUTER

10662 Los Vaqueros Circle, PO Box 3014, Los Alamitos, CA 90720-1264

Editor-in-Chief
Bruce D. Shriver, University of Southwestern Louisiana

Associate Technical Editor
Jon T. Butler, Naval Postgraduate School

Editorial Board
Dharma P. Agrawal, North Carolina State University
Dennis R. Allison, Stanford University
Fletcher Buckley, GE Aerospace
Scott Davidson, AT&T
Richard Eckhouse, MOCO
Michael Evangelist, Andersen Consulting
Scott E. Fahlman, Carnegie Mellon University
John R. Gurd, University of Manchester
Brent T. Hailpern, IBM T.J. Watson Research Center
James H. Haynes, University of California, Santa Cruz
Guy Johnson, Rochester Institute of Technology
Yale Patt, University of Michigan
Guylaine M. Pollock, Sandia National Laboratories
Vincent Y. Shin, MCC
Martha Sloan, Michigan Technological University
Akinori Yonezawa, Tokyo Institute of Technology

Submissions
Send six double-spaced copies of articles and special-issue proposals to Jon T. Butler, Dept. of Electrical and Computer Engineering, Code EC/Bu, Naval Postgraduate School, Monterey, CA 93943-5004, phone (408) 646-3299. For electronic submission, Butler's addresses are j.butler on Compmail II and butler@cs.nps.navy.mil on Internet.

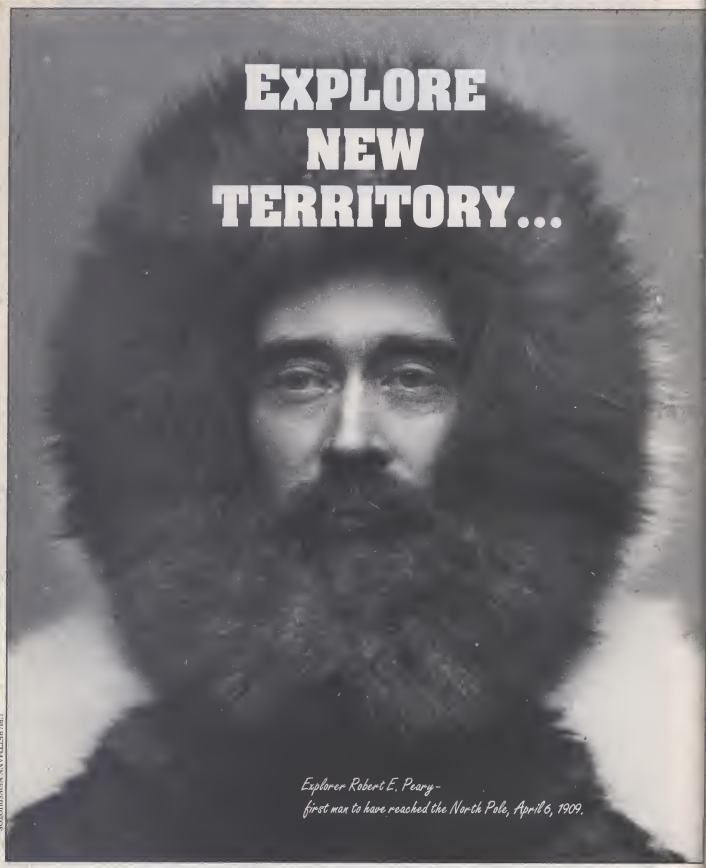
Staff
Publisher: True Seaborn
Editorial Director, CS Magazines: Marilyn Potes
Assistant Publisher: Douglas L. Combs
Assistant to the Publisher: Pat Paulsen

Managing Editor: Marilyn Potes
Staff Editors: Chuck Governale, Steve H. Wilcox, Bob Carlson, Nancy L. Carollo
Contributing Editor: Ware Myers

Art Director: Jay Simpson
Design/Production: Toni Van Buskirk, Janice Garton, Ken Duckworth
Membership/Circulation: Christina Champion
Advertising Coordinators: Heidi Rex, Marian Tibayan

Magazine Advisory Committee
Jon T. Butler (chair), B. Chandrasekaran, Manuel D'Abreu, James Farrell III, Joe Hootman, Anil Jain, Sushil Jajodia, Ted Lewis, True Seaborn, Bruce D. Shriver, Pradip K. Srimani, John Staudhammer

Publications Board
Sallie Sheppard (chair), Dharma P. Agrawal, Victor Basili, P. Bruce Berra, J. Richard Burke, Jon T. Butler, J.T. Cain, B. Chandrasekaran, David Choy, James Cross, Manuel D'Abreu, James J. Farrell III, Joe Hootman, Sushil Jajodia, Glen G. Langdon, Ted Lewis, Ming T. Liu, C.V. Ramamoorthy, True Seaborn, Bruce D. Shriver, John Staudhammer, Harold Stone, Steven L. Tanimoto, Joseph Urban, Ben Wah, Ron Williams

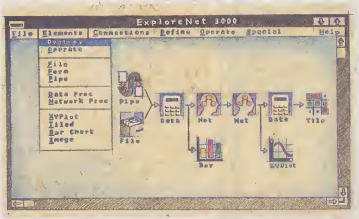


Microsoft Windows and Excel are trademarks of Microsoft Corp. Lotus 1-2-3 is a trademark of Lotus Development Corp. dBase II is a trademark of Ashton-Tate

Order Now
Save over $1000 on Training and Support

Find Hidden Answers.

EXPLORENET 3000™ is advanced neural network software for the PC that helps you find hidden answers in your data. You will experience the ease of developing your application within Microsoft Windows™ 3.0 environment (compatible with over 3,000 other programs on the PC/AT). And best of all, you can develop applications without programming! ExploreNet 3000 applications are 100% compatible with HNC's ANZA Plus™ coprocessors.



File Module

Support fixed and variable length ASCII and binary files, which can be generated by 99% of all programs. For instance, output from such popular programs as Lotus 1-2-3™, dBase II™ and Excel™ can be used without modification.

Data Transformation Module

Perform pre-and post-processing operations by combining functions from a library of over 50 routines. Supports scalar, vector and matrix operations.

Pipe Module

Receive input from and output to thousands of other programs including expert systems, databases and applications programs running on the PC under MS-Windows.

Display Modules

Watch your network train and learn by monitoring the results using six different display options: Plots, Tiles, Images, Forms and Charts (pie charts and bar charts).

Network Module

Tailor any of the 18 well-known neural networks to your application, or modify the architecture of one of the three experimental networks to your specifications.

ExploreNet 3000 will get you up and running faster than any other neural network product on the market—we'll back it with a 30-day money-back guarantee.

Call 1-800-HNC-EXPR

HNC™

5501 Oberlin Drive
San Diego, CA 92121
(619) 546-8877
Fax (619) 452-6524

Reader Service Number 2

NANYANG TECHNOLOGICAL INSTITUTE
SINGAPORE

(To be reconstituted and renamed the Nanyang Technological University in 1991)

GINTIC INSTITUTE OF CIM

The **GINTIC** Institute of CIM is a national Computer-integrated Manufacturing research institute and technology transfer centre set up in the Nanyang Technological Institute to develop local expertise in CIM applications and development by working with industrial partners through cooperative R&D, technical consultancy and the training of CIM specialists.

Applications are invited for the following appointments:

TEACHING ASSOCIATES

FUNCTIONS:
To teach in the Master of Science (CIM) course, supervise the postgraduate students in their project work and carry out research in related areas.

REQUIREMENTS:
Candidates should have a PhD degree in an Engineering discipline or Computer Science/Technology. Relevant industrial experience and expertise in the following areas would be an advantage:

a) Management aspects of CIM
b) CAD/CAM
c) Systems Design
d) Modeling Simulation and Control
e) Electronics Design Automation
f) Electronics Manufacturing Automation
g) Computer Networks
h) Flexible Manufacturing Systems
i) Production Design and Process Planning
j) Intelligent Knowledge Based Systems (IKBS)
k) Manufacturing and Automation

Gross emolument ranges for:

Teaching Associates:
S$53,160 - S$64,200 per annum
Senior Teaching Associates:
S$58,680 - S$100,310 per annum
Principal Teaching Associates:
S$88,650 - S$122,870 per annum

(US$1 = S$1.77 approximately)

The point of entry to the salary ranges stated above will depend on the candidate's qualifications and experience. Other benefits include annual leave, provident fund and medical benefits.

Candidates wishing to be considered should write to:

The Registrar
NANYANG TECHNOLOGICAL INSTITUTE
Nanyang Avenue, Singapore 2263

giving their curriculum vitae and the names and addresses of three referees.

Why we should care about standards

Sizes, shapes, and formats are standardized out of necessity. Without standards, domestic and international trade would be incomprehensible. Standards are developed when people recognize that they have common, recurring problems, and they soon become a way of life that most of us take for granted. They affect our daily lives, often without our awareness. This is especially true in the world of computers, but here it is widely recognized that international standards are one of the best paths to true global competitiveness.

Incompatibility of plug-in units, communications protocols, storage media or software can "lock" computer users into a choice of equipment, services, and supplies that may hurt them in the long run. Enlightened builders of computer-based systems and software want their products to be judged on their capabilities. They recognize that locking users into closed systems (that is, those that do not support standard interfaces) is ultimately a losing strategy.

Increasingly, researchers, administrators, service providers, and organizations depend upon the availability of data in electronic form. The enormous investment in electronic databases motivates users to seek ways to preserve their investments in data and software when computer-based systems are replaced. Where local copies of data are not possible, remote-access to databases is often required. When new businesses are formed or older businesses seek new opportunities, standards can reduce the risks associated with the development of new, information technology-based products and services. These are just a few examples of the forces that have created today's strong interest in information technology standards.

Formal standardization activities in information technology began about twenty-five years ago. Today, both vendors and users are promoting standards for "open" systems. These standards are intended to allow computers manufactured by different vendors to communicate easily. Software based on standard languages and standard interfaces to operating systems, using standard utilities, should be able to run on equipment from different vendors. Users trained on systems supporting these standards should be able to readily use other systems built on the same conventions.

The potential benefits from information technology standards are enormous. Large organizations with big investments in computer equipment and peripherals were the first to recognize this fact. Small, aggressive computer vendors soon saw the opportunity to break into markets previously dominated by a few, large computer companies. Now, even the largest companies see that support for standards is essential to doing business in the global marketplace.

About a year ago, the Board of Telecommunications and Computer Applications of the US National Research Council held a workshop on standards. The impetus for the workshop was a concern that the standards-setting process in the United States is excessively slow and cumbersome for this era of international competition in high technology industries. The workshop report, *Crossroads of Information Technology Standards*, presents the conclusions and recommendations of that meeting.

While several of the workshop recommendations addressed

MESSAGE

US national issues, a number of conclusions can be applied worldwide:

- Those interested in global trade should contribute to and participate in the standards developing process, understand international developments, and support globally accepted directions that would greatly facilitate an efficient and free marketplace.
- The objective of common, global standards achieved through the standards developing bodies should be supported, while the use of in-country, protectionist standards should be opposed.

Developing countries, new companies, and others who are not encumbered with large investments in older technology, have a lot to gain from standards, too. By using widely recognized standards in the design and acquisition of new systems, it is possible to avoid many of the troublesome and expensive problems encountered by those who invested long ago in hardware and software with unique, and thus incompatible, interfaces.

Today, within national, regional and international voluntary standards development organizations, participants are aggressively developing standards in areas such as computer communications, database management, computer graphics, programming languages, and operating systems. These standards promise to reduce costs and increase productivity, while maintaining an open international market for software and services.

The IEEE Computer Society sponsors a large and growing standards program. From a handful of volunteers working on computer terminology in the late 1970s, our program has grown to several thousand people involved with over 190 standards, working groups, and studies. Areas addressed include software engineering, local area networks (the "802" LAN series), microprocessors (Futurebus, for example), operating systems (Posix), design automation (VHDL), computer security, and programming languages (Pascal, for example).

Our objective is to provide a forum for professionals to develop needed, high-quality technical standards to serve worldwide providers, users, and implementors of information technology. All of our working groups are open. Anyone can attend the meetings; we have participants from and hold meetings in many countries around the world. Many of the standards developed under Computer Society sponsorship are adopted by international standards organizations. The international nature of standards development cannot be overemphasized. Narrow, parochial views of standards as a national enterprise, with protectionist overtones, can only lead to diminished competitiveness in the global marketplace.

If you are interested in learning more about the IEEE Computer Society's international standards programs, contact any of the society's offices (see p. 136 for addresses).

Helen M. Wood, Computer Society President

The George Washington University
WASHINGTON DC

DEAN

School of Engineering and Applied Science

Applications and nominations are invited for the position of Dean, School of Engineering and Applied Science.

The Dean is the chief executive officer of the School and is expected to provide vision and leadership in advancing the interests of the school. The new Dean will be a person who can forge strong cooperative working relationships with faculty and university administrators, and will facilitate the generation of, and support of, programs with key government and business entities.

Candidate should possess the following qualifications:

- An earned doctorate in engineering or science-related field
- A distinguished record as a scholar showing commitment to teaching and research
- Demonstrated administrative ability
- Effective interpersonal and communications skills
- Demonstrated ability in obtaining sponsored research support from government and industry
- Ability and interest in fund-raising activities
- Activity in scholarly and professional organizations
- Commitment to continuing and cooperative education

The School of Engineering and Applied Science is one of eight schools within The George Washington University, located in the Foggy Bottom section of Washington, D.C., close to the White House, Kennedy Center and other prominent national monuments. It consists of the Departments of Civil, Mechanical, and Environmental Engineering; Electrical Engineering and Computer Science; Engineering Management; and Operations Research. Within the purview of these departments, several off-campus graduate teaching and research programs are offered. The School will be a major component of the university's new Northern Virginia research/education center.

Application materials should include a cover letter summarizing outstanding qualifications for the position and a current resume. Review of applications and nominations will commence immediately and continue until the position is filled. The intended starting date for the position is July 1, 1991. Salary and benefits are competitive. Send materials to:

Engineering Dean Search Committee
Attention: Ms. Cleo Graves
Room 801, Rice Hall
The George Washington University
Washington, DC 20052

The George Washington University is an equal opportunity/affirmative action employer

THE BEST USE
IN TOWN. NOW
SCREEN N

©1990 Sun Microsystems, Inc. *Sun Microsystems and the Sun logo are registered trademarks of Sun Microsystems, Inc. OPEN LOOK is a trademark of AT&T. All other products or services

R INTERFACE PLAYING AT A EAR YOU.

The OPEN LOOK™ user interface. It's a real hit with independent software vendors, in-house developers and end users. In fact, over 300 applications are in development today. By people like Lotus,® INFORMIX,® Island Graphics,® Interleaf,® and Frame.® And it's the most popular front end to UNIX.® For a number of reasons.

First of all, it makes UNIX easy to use. Because there are no complicated UNIX commands. It also looks better than any other interface. From its icons to its 3D elements. And makes users more efficient. For example, our drag and drop feature gives them a simple, intuitive way to move files around the desktop. Our push-pin icon makes it even easier to use. And OPEN LOOK gives users the same interface across multiple platforms, so they learn it once. And enjoy access to a huge range of network resources.

As a developer, you'll see it's also the easiest to work with. Because it's part of OpenWindows,™ a complete development environment. With the tools you need to create applications faster than ever. And ready-made features, like our DeskSet™ graphical productivity tools, that you can give users right away.

Of course, the business reasons to choose OPEN LOOK are just as strong. OPEN LOOK is the standard interface of AT&T's UNIX System V.4, so it's included at no charge. And it will run on over 20 platforms, including DEC,® HP,® and IBM.® Since it's portable across multiple platforms, you only write your application once. Which saves thousands of man-hours. Finally, with OPEN LOOK, you have the full support of a company that leads the workstation industry in worldwide shipments.*

We've put together a videotape that shows you exactly what OPEN LOOK is all about. Just call us at 1-800-624-8999 (ext. 2068), and we'll send you a free copy.

Then find a nice comfortable seat close to your screen. Because the closer you look, the better we get.

sun microsystems

mentioned are identified by the trademarks or registered trademarks of their respective companies or organizations. *Source, International Data Corporation, 1990. 36.3% market share.

LETTERS TO THE EDITOR

Mea culpa: Magazine misses on music . . .

To the Editor:

While at Bell Labs in the 1960s, I was irked by the hokeyness of being photographed for an AT&T ad ("Bassoon tooter tutors computer") with four other people pretending they were playing instruments (incompatible with the music that I just happened to have brought with me, which was visible). With the appearance of the September 1990 issue of *Computer*, I now find someone appearing to conduct [cover photo: "Formal Methods: Prelude to Virtuoso Software"]. It is immediately evident that the would-be orchestral score is the women's choral reduction for "Carmina Burana" by Carl Orff, conducting from which would be silly in full dress. This situation seems to be something similar to what W.S. Gilbert's Pooh-Bah described in "The Mikado" as "Merely corroborative detail intended to give artistic verisimilitude . . ."

Peter G. Neumann
SRI International

. . . Another small point: In choruses with which I have been affiliated (choral singing has been my hobby for nearly 40 years), those that go in for formal attire require tuxedos for the men in the chorus, while the conductor wears white tie and tails. I do not recall ever working with a conductor who wore white gloves.

Again, these mistakes are not serious, and I had fun working them out. Thanks for the entertainment.

Wallace B. Riley
San Francisco, Calif.

. . . Was it just the handiest score available, or have I missed a subtle message concerning formal methods?

Stephen M. Lott
Washington, DC

. . .Fold-up music stands are not generally used by conductors.

Anonymous
(comment on reply card)

. . . scores low in history . . .

To the Editor:

I was thoroughly disappointed to see in the introduction to the May *Computer* [Joseph Boykin and Susan J. LoVerso, "Guest Editors' Introduction: Recent Developments in Operating Systems," pp. 5-6] the statement, "For example, virtual memory, first shown on the Atlas system in 1959, did not become available in commercial systems until 1972, when IBM released it as part of its general product line." How could you not recognize the B5000, which had virtual memory 10 years previously, as a commercial system? The B5000 was the forerunner of the Unisys A Series, which is still a modern architecture compared to IBM's line of 370 descendants.

Ian Joyner
North Turramurra, Australia

Ian Joyner is correct. Guest Editors Boykin and LoVerso have since explained that by "commercial system" they meant "business system." This was not understood by the copyeditor, who, in recasting the sentence, compounded the error. —Ed.

. . . and fails affiliations

To the Editor:

Daniel E. Geer, Jr., one of my coauthors for "Project Athena as a Distributed Computer System" in the September *Computer*, was listed as being with the Massachusetts Institute of Technology. He was with MIT at the time the article was submitted, but when the article was published, he was an employee of Digital Equipment Corporation.

Please publish a statement to that effect to correct any misconceptions.

George A. Champine
Digital Equipment Corporation

Computer welcomes your letters. Send to Letters Editor, *Computer*, 10662 Los Vaqueros Circle, PO Box 3014, Los Alamitos, CA 90720-1264.
All submissions are subject to editing for style, length, and clarity.

Master of Software Engineering at Carnegie Mellon University

Reply to MSE Admissions Coordinator
CMU / SEI
Dept. C
Pittsburgh, PA 15213-3890

(412) 268-7713

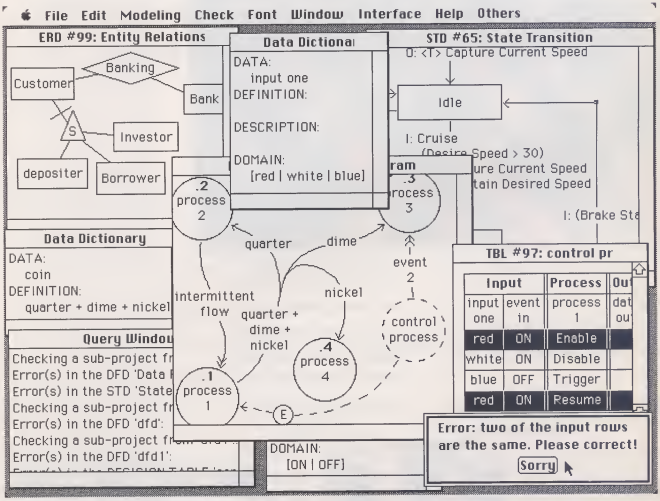
Reader Service Number 4

TurboCASE™: the affordable MAC CASE tool with power to spare.

"TurboCASE is the most usable CASE tool on the market. You can use it right away and be productive right away." says a TurboCASE user in the Feb. 20, 1990 issue of ***MacWEEK****.*

New Version 2.0:

- Structured Analysis (with Real-Time extensions).
- Data Modeling.
- Object Oriented Analysis.



StructSoft, Inc. 5416 156th Ave. SE, Bellevue, WA 98006.
Tel: 206-644-9834 Fax: 206-644-7714

BEFORE OSF/MOTIF AND TELEUSE, THERE WAS A WORD FOR DEVELOPING GUIS.

Building user interfaces for window-based applications on high-powered workstations is pure hell for programmers. Since the user interface generally comprises about 60 percent of the total code for an interactive application, it's also a very time-consuming and costly process. □ Now, however, there's TeleUSE,™ the industry's most advanced user interface management system (UIMS). This integrated set of interactive tools allows programmers to rapidly prototype, design, implement, evaluate and debug standardized graphical user interfaces for OSF/Motif, independently of

the application code. It's a dramatic productivity breakthrough for anyone developing OSF/Motif applications. □ With the TeleUSE graphical layout editor and dialogue manager developers can focus more on the design, esthetics and viability of the user interface rather than writing code. And cut the development time by 50 to 90 percent. □ For complete details on TeleUSE, call TeleSoft at (619) 457-2700. Or write TeleSoft, 5959 Cornerstone Court West, San Diego, California 92121.

NO MORE WRITING THOUSANDS OF LINES OF CODE TO CREATE GUIS FOR OSF/MOTIF. LET TELEUSE, THE INDUSTRY'S HOTTEST UIMS, AUTOMATE THE PROCESS FOR YOU. AND WATCH YOUR PRODUCTIVITY TAKE OFF!

Using a WYSIWYG approach, software developers use the TeleUSE layout editor to actually paint the static screen layout on a graphics workstation using building block interface objects such as buttons, scroll bars, pop-up menus, and dialogue boxes.

TELESOFT

PROGRAMMED FOR PRODUCTIVITY

Reader Service Number 6

Multiprocessor Algorithms for Relational-Database Operators on Hypercube Systems

Ophir Frieder
George Mason University

As databases expand and applications become more diverse, demands on computational engines supporting database processing increase. With appropriate algorithms, commercially available hypercube systems can meet the demands.

Exploiting parallelism in database processing has been a research goal since the early 1970s. Originally, special-purpose architectures were developed to provide the computational and input/output bandwidth needed for database processing. More recently, many researchers have relied on commercial multiprocessor and local area networked systems by such vendors as Sequent, Tandem, BBN, DEC, and various hypercube system manufacturers to improve database processing performance.

This tutorial focuses on hypercube interconnected architectures as a computational engine for relational-database processing. Like other architectures with distributed memory and resources ("shared-nothing" architectures[1]), hypercube systems can support the high I/O bandwidth required for database processing. However, unlike the other architectures, hypercubes are scalable to thousands of nodes. For example, NCube Corporation currently manufactures hypercubes comprising up to 8,192 nodes. These engines can provide large-scale concurrency for both interquery and intraquery processing[2] and are well suited for such computationally intensive processing as protocol verification using database technology.[3]

After reviewing hypercube systems, this tutorial briefly highlights several implementations of the many currently available hypercube systems and comments on their potential performance in evaluating relational database operators. All algorithms assume that the relevant data are memory resident.

The hypercube multicomputer

A hypercube graph is an n-dimensional Boolean cube Q_n defined as a cross product of the complete graph K_2 and the $(n-1)$-dimensional Boolean cube Q_{n-1}, with $Q_1 = K_2$. In an architecture based on hypercube interconnection, each node is connected

0018-9162/90/1100-0013$01.00 © 1990 IEEE

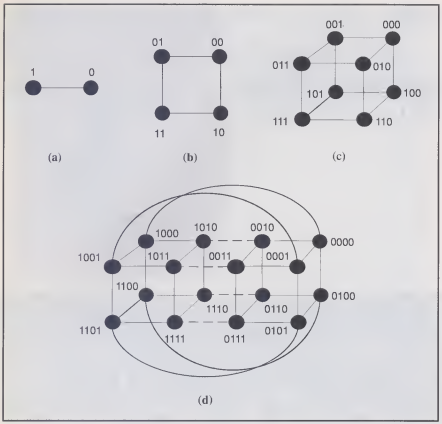


Figure 1. Various hypercube configurations: (a) one-dimensional, (b) two-dimensional, (c) three-dimensional, (d) four-dimensional.

(or adjacent) to each of its $n = \log_2 N$ neighbors, where N is the number of nodes. For example, in a four-dimensional cube Q_4, node 0000 is adjacent to nodes 0001, 0010, 0100, and 1000. Figures 1a through 1d illustrate the communication paths of one-dimensional (two-node), two-dimensional (four-node), three-dimensional (eight-node), and four-dimensional (16-node) hypercube systems. Note that each system consists of $N = 2^n$ nodes, with n being the cubical dimension of the system. We assume that the node address bits are numbered 0 to n-1, with the leftmost bit (bit 0) being the most significant. Existing hypercube machines include Caltech's Cosmic Cube,[4] Intel's iPSC/2, and NCube's NCube/10.

Nodes communicate by sending messages in packets. Packet size varies, but protocol imposes a maximum. Packets, which in database processing contain tuples, can be sent between any two nodes in the system, possibly being routed through intermediate nodes. In current hypercube systems, typical internode communication times are on the order of microseconds.[5]

Synchronization among nodes can be achieved either through hardware support or strictly through software. In software synchronization, a receiving node "blocks" until a message arrives. Thus, the arrival of a message synchronizes the two nodes. This blocking send/receive technique can be generalized to synchronize all nodes within the system. A possible synchronization algorithm for hypercube architectures is based on a message-sending ordering technique called recursive halving, which I will describe in a later section.

Other hypercube systems use global hardware lines to synchronize the processors. A global wired-AND or wired-OR line is connected to a number of nodes, and each node has a local input. The global line value is the logical AND'ing or OR'ing of the local inputs. Synchronization is maintained by monitoring the changes in the global line value. For example, if a "global AND" line is used, all nodes maintain their local input value at false throughout the execution of their local task. A global line value of true implies that all nodes terminated local execution. Similarly, wired-OR lines can be substituted for wired-AND lines by reversing the local line values. The algorithms presented in this article assume the availability of global synchronization lines. These lines need not be supported by hardware but may be provided logically by software.

Other scalable "shared-nothing" architectures include mesh-based multicomputers and ring-based local area networks. Each hypercube node requires more communication ports than the nodes making up meshes and rings, but the hypercube interconnection significantly reduces the maximal communications diameter as compared with meshes and rings. The maximal communications diameter in a hypercube comprising N nodes is $\log_2 N$ as compared with $N/2$ and $\sqrt{N}$ in similarly sized rings and meshes, respectively.

Hypercube architectures as database engines

Database processing requires vast I/O and data-access bandwidth and significant computational resources. Hypercube systems provide all three. By horizontally partitioning relations (see sidebar on pp. 16-17), across a parallel I/O structure like the disk-per-node I/O subsystem in the Intel iPSC/2, a relation can be read or written in parallel, if appropriate synchronization primitives are available. Thus, currently available hypercube systems can remedy the known I/O bottleneck of database processing.

The degree of I/O parallelism depends on the partitioning scheme and the data values relevant to the query. Common horizontal partitioning techniques include random, round-robin, hash-based, range-based, and user-specified distribution of tuples across the sites. Both the hash-based and the range-based partitioning approaches allocate the tuples according to the hashed or actual value of a set of specified attributes. A round-robin partitioning scheme evenly distributes the tuples across the sites.

The Gamma Project[6] is a hypercube database engine initially developed for a ring-based multicomputer. The system currently runs on an Intel iPSC/2 hypercube comprising 32 80386-based nodes with a disk drive per node. Gamma exploits the available parallel I/O capability by horizontally partitioning the relations across the disk drives. The relational-database operator algorithms presented in this tutorial differ from those developed for the Gamma project[7] in that they are not independent from interconnection

topology, and hence are actually optimized for the hypercube interconnection topology.

Database performance is enhanced whenever the variance in the data distribution across the various processing sites is low. Poorly distributed relations result from individual database operators that favor a particular value, for example, a selection. Independent of the data organization, data skew is likely to result during the processing of some user queries. For example, consider the pathological case of a 50-processor system on which a population database is partitioned according to states: A range-based horizontal partitioning scheme based on the STATE attribute is used, resulting in each processor accessing data for only a single state. A query interested in information about only the state of Michigan yields data at only a single processor. Without redistributing the data, 49 of 50 processors remain idle for the duration of the query.

Dynamic data redistribution (on-the-fly data reorganization) preceding each multiscan operator in a query tree has been proposed to guarantee a near-even workload across the processors. A *multiscan operator* is any operator in which the processing of an individual tuple involves comparing its attribute value(s) against other tuples. For example, the Select is not a multiscan operator since the relevance of the tuple is independent of any other tuples. However, both the Join and Project are multiscan operators because the relevance of each tuple depends on the values of other tuples: unique value in the case of Project, and a similar value in the other joining relation in the case of Join (see sidebar).

Baru and Frieder[8] have demonstrated the reduction in time resulting from executing a nested-loop Join on a hypercube instead of a special-purpose, bus-based architecture. The study also demonstrated that data redistribution on a hypercube system as part of the Join can be achieved with low overhead. A 16-node hypercube system using dynamic data redistribution achieved roughly a 15- to 80-percent reduction in the processing times of various Join computations. The exact savings depended on the degree of skew in the data distribution.

As databases expand in size and applications using databases become more diverse, the computational engines supporting database processing must likewise continue to improve. Currently, hypercube systems comprising more than 8,000 processors are available commercially.

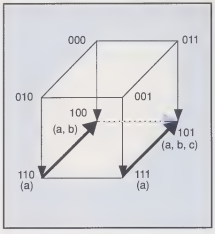


Figure 2. Aggregation example.

Thus, for at least the near future, hypercube systems meet the computational demands of database processing.

Hypercube database algorithms: Uniscan operations

A query tree is a partially ordered sequence of operators initiating at the leaves, with the result of the query obtained on termination of root execution. The output relation of the child operator is the input relation of the parent operator. The relations accessed by the leaf nodes are called base relations and typically are physically stored in the database. All computed relations, except for the final output, are called *intermediate relations*.

The relational operators can be viewed as being in one of two categories: *uniscan* and *multiscan* operators. Both Join and Project are examples of a multiscan operators since each tuple, in turn, is compared against a set of tuples. However, both Select and the aggregation operators are uniscan operators because the processing of each tuple is independent of the processing of any other tuple.

The Select operator. In optimized query trees, the Select operators are typically located near the leaf levels of the query tree. Thus, in a selection on a horizontally partitioned relation, it is common to assume a *uniform* distribution of tuples across the nodes. As Select is a uniscan operator, each tuple can be processed independently. Hence, computing a Select in parallel requires each node to read its resident tuple set and, for each tuple read, compare the tuple attribute value against the desired value. If the node detects a match, it keeps the tuple.

Aggregation operators. Scalar aggregation operators are an extension to the relational algebra and include such uniscan operators as Max, Min, Count, and Average. In parallel systems, aggregation is generally performed in two phases. In the first phase, each node computes its local aggregate value. Tuples are accessed as described for the Select operator. In the second phase, the global aggregate value is computed by combining all the local values at the final destination or target node.

On a hypercube the global aggregation phase, phase 2, takes n steps and is based on a common technique called recursive halving. As the communications diameter of the hypercube is n, n steps represent the minimal number of steps required for the scalar aggregation, since relevant data can reside on any node in the cube. In the kth step, $k = 0$ to $n-1$, nodes whose leftmost k address bits equal the leftmost k bits of the target address receive the intermediate aggregate value from the nodes that differ in address from the target in the kth bit.

Figure 2 illustrates an eight-node cube where node 5 (101) is the target node. Target node designation is specified in the query. In step a ($k = 0$), each of the nodes 0 (000), 1 (001), 2 (010), and 3 (011) sends its value to nodes 4 (100), 5 (101), 6 (110), and 7 (111), respectively. The receiving nodes, indicated with the letter a, compute the new aggregate values. These new values are used in the second step, step b ($k = 1$), where nodes 6 (110) and 7 (111) send their values to nodes 4 (100) and 5 (101), respectively. Receiving nodes are marked by the letter b. Once again, the receiving nodes compute the new values. Finally, in step c ($k = 2$), node 4 (100) sends its value to node 5 (101), the target node marked by the letter c, which computes the final result.

Sometimes the user may wish to aggregate values by categories. For example, in a population database for the United States, a user may want the average age of the population in each state. In such cases, the aggregation is performed by repartitioning the data according to attribute values (cat-

(Continued on p. 18)

Relational database primitives and nomenclature

Many database-management systems are available today. A majority of the most recently developed systems are based on the relational-database model, in which the operators available to the user operate on relational structures. An attribute is any symbol from a finite set $\mathfrak{L} = \{A_0, A_1, A_2, ..., A_n\}$. A relation $\mathfrak{R}$ on the set $\mathfrak{L}$ is a subset of the Cartesian product of dom $(A_0) \times$ dom $(A_1) \times$ dom $(A_2) \times ... \times$ dom (A_n), where dom (A_i) is the domain of A_i. $R[A_0\ A_1\ A_2\ ...\ A_n]$ represents $\mathfrak{R}$ on the set $\{A_0, A_1, A_2, ..., A_n\}$ and is referred to as the schema of $\mathfrak{R}$. In $R[A_0\ A_1\ A_2\ ...\ A_n]$, each column A_i is called an *attribute* of R, and is denoted as R.A_i. Each row of R, namely a *tuple*, is designated by $<a_0, a_1, a_2, ..., a_n>$, where $a_i \in$ dom (A_i). The value of attribute A_i of tuple $x \in R$ is denoted as $x[A_i]$. Similarly, if tuple $x \in R$, then $x[W]$ is the value of the attributes of attribute set W in tuple x.

For illustration, assume the relations shown in Table A. Relation EHW has three attributes — EHW.Employee_No., EHW.Height, and EHW.Weight — while relation EA has only two attributes — EA.Employee_No. and EA.Age. Each relation consists of 16 tuples.

Three of the more common operators in the relational model include the Select, Project, and Join. These three operators are formally defined as follows:

Select – The selection on R[XYZ], denoted as $\sigma_{A=a}$ (R), is defined by

$$\sigma_{A=a}(R) = \{x \mid x[A] = a, x \in R\},$$

where A is an attribute of R.

Project – The projection on R[XYZ], denoted as π_A (R), is defined by

$$\pi_A(R) = \{x[A] \mid x \in R\},$$

where A is a set of attributes of R.

Join – The Join of two relations R[XYZ] and S[VWX], denoted as

R[XYZ] |×| S[VWX], is defined by
R[XYZ] |×| S[VWX] = $\{x \mid x[VWX] \in S$ and $x[XYZ] \in R\}$,

where V, W, X, Y, and Z are a disjoint set of attributes. If no common joining attributes exists, the Join of R and S, is the Cartesian product of R and S.

Using the above relations,

List all the employees who are 72 inches tall.

E72W [Employee_No. Height Weight]
= $\sigma_{EHW.Height = 72}$ (EHW) =

Selecton example:

Employee_No.	Height	Weight
101	72	195
303	72	180
801	72	187

Table A. Sample relations EHW and EA.

Employee Height and Weight Relation (EHW)			Employee Age Relation (EA)	
Employee_No.	Height	Weight	Employee_No.	Age
101	72	195	101	31
106	69	141	106	26
115	70	182	115	40
210	64	108	210	25
211	74	185	211	45
301	68	172	301	37
302	71	201	302	52
303	72	180	303	34
304	70	165	304	43
454	62	180	454	35
531	64	125	531	29
640	73	212	640	32
801	72	187	801	55
802	71	198	802	33
803	73	170	803	28
804	67	210	804	34

Enumerate all unique heights of the employees.
UH [Height] = $\pi_{EHW.Height}$ (EHW) =

Projection example:

Height
62
64
67
68
69
70
71
72
73
74

Find the weight and age of all the employees who are 72 inches tall.

E72W [Employee_No. Height Weight] |×| EA [Employee_No. Age] =

A Join example:

Employee_No.	Height	Weight	Age
101	72	195	31
303	72	180	34
801	72	187	55

In the project operator, only one <72> tuple exists. That is, the duplicate <72> tuples generated were eliminated.

A common extension to the relational database is the aggregation operation. The operator, denoted by $agg_{f(x)}(R)$, computes a global aggregate function on attribute set X of the relation R. Ex-

amples of an aggregate function, $f(x)$, include Sum, Max, Min, and Average. For example, the total sum of the EHW.Height is

$$agg_{sum(EHW\ Height)}(EHW) = 1{,}112.$$

Note that the above is not equivalent to the sum of $\pi_{P.B}(P)$,

$$agg_{sum}(\pi_{EHW.Height}(EHW)) = 690.$$

Additional operators such as Theta-Join, Union, Difference, and Renaming have also been incorporated in many existing database-management systems. For a complete treatise on the relational-database model, see D. Maier, *The Theory of Relational Databases* (Computer Science Press, Rockville, Md., 1983).

Database Partitioning

In distributed and parallel database systems, two common relation-distribution schemes are horizontal and vertical partitioning. A relation $R[A_0\ A_1\ A_2\ ...\ A_n]$ is horizontally partitioned across multiple sites S_i, $0 \le i \le k$, denoted $H_{i,R[\ A_0\ A_1\ A_2\ ..\ A_n]}$, $0 \le i \le k$, if $x \in H_{i,R[\ A_0\ A_1\ A_2\ ...\ A_n]} \rightarrow x \in R[\ A_0\ A_1\ A_2\ ...\ A_n]$ and $\bigcup_i (H_{i,R[\ A_0\ A_1\ A_2\ ..\ A_n]}) = R[A_0\ A_1\ A_2\ ...\ A_n]$. Informally, the tuples of R are partitioned so each site contains a possibly empty subset of the tuples of the original relation. A relation $R[A_0, A_1, A_2, ..., A_n]$ is vertically partitioned across multiple sites S_i, $0 \le i \le k$, denoted $V_{i,R[X]}$, $0 \le i \le k$, if $[A_0, A_1, A_2, ..., A_n] \supset X_i$, $\pi_{Xi}\ (R) = V_{i,R[Xi]}$, and $V_{i,R[Xi]}\ |x|\ V_{j,R[Xj]}\ |x|\ ...\ |x|\ V_{k,R[Xk]} = R[A_0\ A_1\ A_2\ ...\ A_n]$. Simply stated, vertical partitioning distributes the relation by attributes so each site contains a number of attributes from the base relation. The Join of the partial relations at the individual sites yields the original relation: The Join composition is lossless.

The data-partitioning scheme significantly affects the total query-processing time. If an application requires that at each site only tuples whose attribute values fall within a specified range are present, but that the entire contents of the resident tuples be available, then horizontal partitioning should be used. Horizontally partitioning the relation reduces the volume of data that must be processed at each site without incurring any interprocessor communication. If vertical partitioning is used, additional Joins would be required to obtain the nonresident attribute values. The additional Joins result in both added interprocessor communication and increased computation.

If an application requires at each site only a subset of the tuple's attributes, then vertical partitioning should be used. Vertical partitioning of the relation reduces the volume of resident data and makes unnecessary the projection of the desired attributes from the original relation. Horizontal partitioning requires a projection on the union of all the local data sets and results in additional interprocess communication.

To see the advantages of each relation-partitioning scheme, assume that a company has an Employee_No., Height, Weight (EHW) database as in Table A and that it has one geographically distant division. Further assume that the employee numbers of all employees at site 1 range from 100 to 499, whereas all employees at site 2 have employee numbers ranging from 500 to 899. As all the information concerning each employee is required at the employee's local site, horizontal and not vertical partitioning of the EHW relation is appropriate. Table B shows a logical horizontal partitioning of the EHW.

Now consider the hypothetical situation that at each site there are freight elevators with maximum weight limitations and storage rooms with low ceilings. For each employee, only weight or height, respectively, is needed to determine whether he or she can join the people or cargo already in the elevator or enter the room. In this case, each local database should be vertically partitioned, as shown inTable C.

This simple example demonstrates that a composition of both horizontal and vertical partitioning can reduce the processing demands.

Table B. A logical horizontal partitioning of the EHW.

$H_{1,EHW[Employee\_No.\ Height\ Weight]}$

Employee_No.	Height	Weight
101	72	195
106	69	141
115	70	182
210	64	108
211	74	185
301	68	172
302	71	201
303	72	180
304	70	165
454	62	180

$H_{2,EHW[Employee\_No.\ Height\ Weight]}$

Employee_No.	Height	Weight
531	64	125
640	73	212
801	72	187
802	71	198
803	73	170
804	67	210

Table C. A logical vertical partitioning of the horizontal partitioning of EHW.

$V_1,H_{1,EHW[Employee\_No.\ Height\ Weight]}$

Employee_No.	Height
101	72
106	69
115	70
210	64
211	74
301	68
302	71
303	72
304	70
454	62

$V_2,H_{1,EHW[Employee\_No.\ Height\ Weight]}$

Employee_No.	Weight
101	195
106	141
115	182
210	108
211	185
301	172
302	201
303	180
304	165
454	180

$V_1,H_{2,EHW[Employee\_No.\ Height\ Weight]}$

Employee_No.	Height
531	64
640	73
801	72
802	71
803	73
804	67

$V_2,H_{2,EHW[Employee\_No.\ Height\ Weight]}$

Employee_No.	Weight
531	125
640	212
801	187
802	198
803	170
804	210

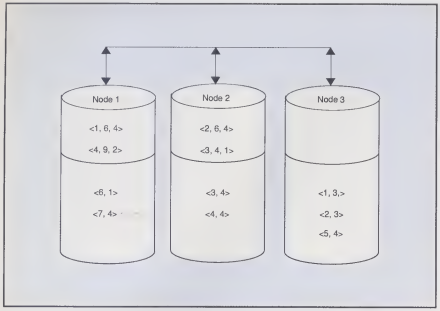


Figure 3. Database partitioning for a multiprocessor Join example.

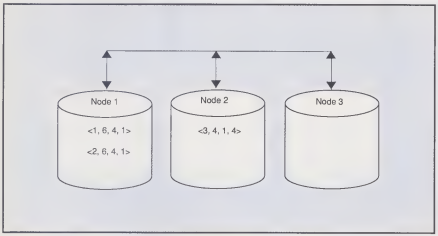


Figure 4. Broadcast-based Join result.

egories), with each node computing the aggregate for the resident categories. I discuss data redistribution according to attribute values in later sections.

Multiscan database operators

Multiprocessor implementations of the multiscan database operators can be classified into two main categories: broadcast-based and bucket-based. The first category, broadcast-based, requires that each node, in turn, broadcast its portion of the smaller relation, R_I (in the case of a two-relation operator), or the relation at hand (unirelational operator), to all the system nodes. All nodes receive the broadcast message (tuple set) and perform the appropriate local computation, which involves the resident tuples and the received packet.

Bucket-based solutions include implementations that rely on sorting and/or hashing techniques. (Numerous extensions and modifications to the basic bucket approach described here appear in the literature.[7,9,10]) We characterize any operator-processing approach as bucket-based if the approach partitions all the data elements involved in the operation into buckets according to their attribute values. Each bucket corresponds to a range of attribute values, and only tuples consisting of attribute values within the given range reside in the bucket.

This approach has an advantage over the broadcast-based approach: Only tuples that are likely to match are compared with one another. However, this approach suffers from the need to repartition both relations for a multirelational operator. Furthermore, when one relation is significantly larger than the other, the communication requirements of the bucket approach result in greater processing time than a broadcast-based Join. The time involved in partitioning both relations instead of only the smaller relation is greater than the savings obtained by eliminating redundant comparisons.

An example illustrates both approaches. Consider a Join of two relations P and T on attribute B, $R[ABCD] = P[ABC] \bowtie T[BD]$, on a three-node multiprocessor where the relations are partitioned as shown in Figure 3. In the figure, P is shown as the top relation. Node 1 contains four tuples, two of P and two of T. Node 2 contains two P tuples and two T tuples, while node 3 has only three T tuples. Clearly the P relation consists of fewer tuples; hence, we refer to it as the smaller relation.

A broadcast-based Join algorithm proceeds as follows. Since P is the smaller relation, each node in turn broadcasts each tuple of P to all other nodes. Thus, node 1 will broadcast <1, 6, 4> and <4, 9, 2>, node 2 will broadcast <2, 6, 4> and <3, 4, 1>, and node 3 will not broadcast any tuples. End-of-tuple transmission is indicated by a default-value broadcast. All nodes monitor the transmission and compute the local Join of the broadcasted tuple value and their local T tuples. Figure 4 illustrates the resulting relation.

Continuing with the example, initially the maximal possible joining attribute(s) range is computed. This range consists of actual attribute values and not necessarily the domain of the joining attribute(s). For example, the range of B of the "combined" relation, R, is 1 to 9, designated attr-ran(R.B) = [1,9]. However, dom(B) may comprise a much greater range, say the set of natural numbers. A tighter bound on maximal attr-ran(R.B) is the intersection of the regions attr-ran(P.B) = [4,9] and attr-ran(T.B) = [1,7], namely maximal attr-ran(B) = [4,7].

Once the attr-ran[R.B] is computed, the corresponding disjoint attribute range is assigned to each processor. Assigning each processor an attribute range is accomplished

either statically or dynamically. In a system using static partitioning, the range attr-ran(B) is partitioned equally across the processors. Such a partitioning scheme results in a skewed processor workload whenever the distribution of attribute values is biased.

A dynamic partitioning scheme nullifies the effects of a skewed tuple distribution. Randomly sampling a small number of the joining tuples as part of the previous operation provides a crude histogram of the data distribution and hence of the processor workload distribution. Using this information, the range attr-ran(R.B) is partitioned to yield a near-optimal workload partitioning of the Join processing across the processors. However, like all dynamic load-balancing algorithms, dynamic sampling introduces some overhead. The precise observed reduction in processing time depends on the skewness of the data and various machine-specific parameters, for example, communications overhead.

The example presented here uses static partitioning. Nodes 1, 2, and 3 are assigned B attribute value ranges 4-5, 6, and 7, respectively. All tuples of both relations are redistributed on the basis of their B attribute value to reside in the proper bucket. When the redistribution terminates, all processors compute the local Join of their P and T tuples. Figure 5 illustrates the resulting tuple distribution for the bucket Join method. Comparison with Figure 4 shows that both schemes produce the identical relation; however, the relation-partitioning differs.

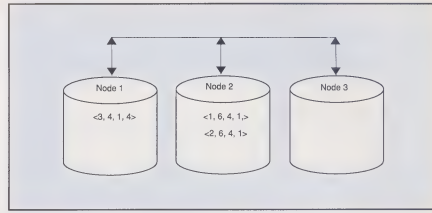


Figure 5. Bucket-based Join result.

The Join operator: A broadcast-based solution

Because broadcast-based solutions reduce communication costs (transmit less data) at the expense of redundant comparisons, ensuring full processor use is vital. Baru and Frieder[8] describe a three-stage, broadcast-based Join that guarantees a nearly even processor workload. First, a dynamic data-redistribution algorithm referred to as *balancing* ensures an even distribution of input tuples. Incorporating the balancing step results in a minimal increase in communication costs but roughly a 35-percent reduction in the overall Join processing time. Once they are evenly distributed, the tuples are sorted.

In the second stage, the Relation Compaction and Replication (RCR) stage replicates the smaller relation R_1, originally stored in a cube of dimension n, so it is replicated in each of the two equal-sized, dimension n–1, logical-cube partitions of the original cube. This primitive's goal is to increase the number of tuples from R_1 stored at each node until the volume of R_1 tuples present at each node is the size of one packet, or until R_1 has been fully replicated at each node. It ensures that packets used in the cycling phase are as full as possible, and that the packet-formation overhead per tuple is minimized for the cycling primitive.

Finally, the cycling primitive sends the tuples of the smaller relation around in a ring. Local Joins of the resident tuples and the pipelined circulated relation are performed simultaneously at each node. Often the design of each node incorporates dual buffers, so the broadcast of packet i+1 can overlap the local computation of packet i.

Assuming multiple independent communications processors, the balancing of two relations simultaneously proceeds as follows. Initially, the local tuple counts of the relations R_1 and R_2 are computed. During each step j ($0 \le j \le n-1$), the nodes whose addresses differ in the jth bit exchange their local-relation R_1 tuple count. The node with the greater number of tuples (if any) then sends the difference of the average tuple count and its own tuple count of tuples to its paired neighbor. Simultaneously, the nodes whose addresses differ in the $((j+1) \bmod n)$th bit balance R_2. The local tuple counts for R_1 and for R_2 are then recomputed, and the next step is initiated. After n steps, all nodes contain roughly the same number of R_1 and R_2 tuples. A more complex balancing algorithm that incorporates the sending-node number can guarantee a difference of one tuple per node, per relation. Once balancing is complete, both relations are sorted locally.

Figure 6 provides a simplified pseudocode description of the single-relation balancing algorithm. Several instructions require explanation. The *send(addr, data)* and *receive(addr, data)* send or receive from nodes whose address is *addr* the tuples designated by the set *data*. *Rotate_right(num, count)* rotates *num*'s binary representation *count* bits to the right. Similarly the *rotate_left(num, count)* instruction in the RCR primitive (Figure 7) rotates the *num*'s binary representation *count* bits to the left. For example, using a four-bit representation of 1 (0001), rotate_left (1, 1) = 2, rotate_right (1, 1) = 8, and rotate_left (1, 2) = rotate_right (1, 2) = 4. Finally, Global AND (*GAND*) is the global synchronization line value obtained by AND'ing all the local synchronization line values. The *ABS*, *XOR*, and *CEIL* instructions are the mathematical absolute value, exclusive OR, and ceiling functions, respectively. The node address bits are numbered 0 to n–1.

The second stage of the broadcast Join, relation compaction and replication, proceeds as follows. Initially the local tuple count of the smaller relation, say R_1, is computed. During each step j ($0 \le j \le n-1$), the nodes whose addresses differ in the jth bit exchange their local R_1 tuple count. RCR is possible if the combined volume (in bytes) of R_1 tuples in each pairing of nodes does not exceed the maximal size of a single packet. When possible, all nodes transmit their tuples to their paired neighbors. While maintaining a sorted order, the local and received tuples are merged, and the new

```
begin { BALANCE }

{ Redistribute data dynamically according to the volume of data, ensuring a
    nearly even workload across the processors.

  PREDEFINED FUNCTIONS:
      FIRST ( k )          Function that moves the first k tuples of a
                             prespecified relation to a stated buffer (send_set)
      XOR (k, j)           Bitwise Exclusive OR of k and j
      rotate_right (k, j)  Rotate the binary representation of k, j bits to the right

  INPUT:                   Number of locally resident tuples (own_count)
                           Local address of the node (X)

  PRIOR TO EXECUTION:      Relation randomly distributed across nodes

  EXECUTION TERMINATION:   Evenly distributed relation across nodes  }

  for j := 0 to n – 1 do begin
    send ( XOR( X, rotate_right ( 1, j+1 ) ), own_count );
    receive ( XOR( X, rotate_right ( 1, j+1 ) ), neighbor_count );

{ Simultaneously equate the tuples across each node pair.      }
    if ABS ( own_count - neighbor_count ) > 1 then begin
        avr := CEIL ( [ own_count + neighbor_count ] / 2 );

        if own_count > neighbor_count then begin
          send_set := FIRST ( own_count – avr );
          send ( XOR( X, rotate_right ( 1, j+1 ) ), send_set );

          loc_tuple_set := loc_tuple_set - sent_set;
          own_count := own_count - | send_set |;
        end
        else begin
          receive ( XOR( X, rotate_right ( 1, j+1 ) ), receive_set );

          loc_tuple_set := loc_tuple_set + receive_set;
          own_count := own_count + | receive_set |;
        end
        end

  end { for }
end { BALANCE }
```

Figure 6. Data-balancing pseudocode.

tuple count is computed. RCR continues until either n RCR steps have occurred or the single-packet limitation is exceeded. Each successful RCR step results in the duplication of the smaller relation. After j successful RCR steps, a copy of R_1 is resident in each of $2^{(n-j)}$ dimensional hypercubes. Figure 7 shows a pseudocode description of the RCR stage.

The final stage, called *cycling*, creates a Hamiltonian cycle within each logical cube partition generated by the RCR primitive, and pipelines the data packets throughout the cycle. Only those address bits not used in the RCR step are considered when forming the cycle. All nodes within each formed cycle have the identical bit values in the *iteration* rightmost address bits. A Hamiltonian cycle can be dynamically generated with reflexive Gray codes, as shown in Figure 8. Executing the pseudocode results in a reflexive Gray code ring in Send_Array. Given any node number, say stored at Send_Array [i], its left and right ring neighbors are the values at Send_Array [$i-1$] and Send_Array [$i+1$], respectively.

We now work through an example. Consider two relations partitioned over an eight-node hypercube, $N = 8$ and $n = 3$. Figure 9 shows the initial distribution of the 24-tuple relation R_2 (upper portion of each bin) and 16-tuple relation R_1 (lower portion of each bin). Initially, the balancing stage is used, resulting in a uniform distribution of both R_2 and R_1. As shown in Figure 10, in steps 1, 2, and 3 a maximum of two, one, and one tuples, respectively, are transferred in a neighbor pairing. The figure illustrates two relations being balanced simultaneously. Assuming a maximum packet limitation of four tuples — because there are two R_1 tuples after balancing at each node — a single RCR step is possible. After the RCR step, two full copies of the smaller relation exist. The first copy of R_1 is partitioned over nodes 0 (000), 2 (010), 4 (100), and 6 (110), while the second copy spans nodes 1 (001), 3 (011), 5 (101), and 7 (111). Note that a sorted order is maintained. Finally, a Hamiltonian cycle is generated in each of the smaller cubes — (000→ 010 → 110 → 100 → 000) and (001 → 011 → 111 → 101 → 001) — and the compacted smaller relation is circulated in both cubes simultaneously. Figure 11 gives the resulting relation.

The Join operator: A bucket-based solution

Nearest neighbor pairing algorithms — for example, the balancing algorithm presented in Figure 6 — can implement bucket-based Joins. Initially all nodes sort their local tuple set of both relations based on the joining attribute values. In bucket-based Join algorithms, instead of computing the average number of tuples between each pair of nodes, at every node-pairing step j ($0 \le j \le n-1$), each node partitions its tuple set into two. The first set S_1 consists of all tuples whose attribute values are assigned to hypercube nodes whose jth bit address is 0. All remaining tuples are placed in S_2. With a sorted tuple set, partitioning the tuples into two sets only requires finding the boundary condition.

During each step j ($0 \le j \le n-1$), nodes

```
begin { RELATION COMPACTION REPLICATION }

{ Replicates a relation partitioned across the nodes of
  Qn into each Qn-1. Via replication, the number of
  nodes required for the later stage, cycling, is halved.
  Replication continues until packet limitation is reached.

  PREDEFINED FUNCTIONS:
    FIRST ( k )          Function that moves the first k tuples
                           of a prespecified relation to a
                           stated buffer (send_set)
    XOR (k, j)           Bitwise Exclusive OR of k and j
    SYNCH (GAND)         Global AND'ing of all local
                           feeds (GAND_loc)
    rotate_left (k, j)   Rotate the binary representation
                           of k, j bits to the left

  INPUT:                 Local address of the node (X)

  PRIOR TO
  EXECUTION:             Relation distributed across all
                           nodes in Qn

  EXECUTION
  TERMINATION:           2^iteration copies of the relation, each
                           partitioned over a disjoint Qn-iteration }
  GAND_loc := true;
  iteration := 0;

  SYNCH( GAND );

  While GAND and ( iteration ≤ n − 1 ) do begin

  { Inform neighbor of own count }
    f_send ( XOR( X, rotate_left ( 1, iteration ) ),
                own_count );
    f_receive ( XOR( X, rotate_left ( 1, iteration ) ),
                neighbor_count );

  { Determine if compaction / replication is possible    }
                if (own_count + neighbor_count) >
                MAX_TUPLES_IN_PACKET
                then GAND_loc := false;

    SYNCH( GAND );

    if GAND then begin  { step is possible   }

      send ( XOR( X, rotate_left ( 1, iteration ) ),
        tuple_set );
      receive ( XOR( X, rotate_left ( 1, iteration ) ),
        receive_set );

      iteration := iteration + 1;
      tuple_set := tuple_set ∪ receive_set;
      own_count := own_count + | receive_set |;
        end
      end { step possible }
  end { while }
end { RELATION COMPACTION REPLICATION }
```

Figure 7. Pseudocode for relation compaction replication.

```
begin  { GENERATE CYCLE }

{ Generate a routing table that yields a reflexive
  Gray code in each of the Qn-iteration that resulted
  from the RCR step.

  INPUT:          Cube dimension (dimension_cube)

  OUTPUT:         Reflexive Gray code routing table  }

  Send_Array [ 0 ] := 0;
  Send_Array [ 1 ] := 1;
  curr_num_nodes := 2;

  For dimen := 2 to dimension_cube do begin

    curr_num_nodes := curr_num_nodes × 2;
    j := -1 ;

    For i := ( curr_num_nodes / 2 ) to
      curr_num_nodes − 1 do begin

      j := j + 2;
      Send_Array [ i ] := Send_Array [ i −j ]
          + curr_num_nodes / 2;
    end;
  end;
end; { GENERATE CYCLE }
```

Figure 8. Cycle-generation pseudocode.

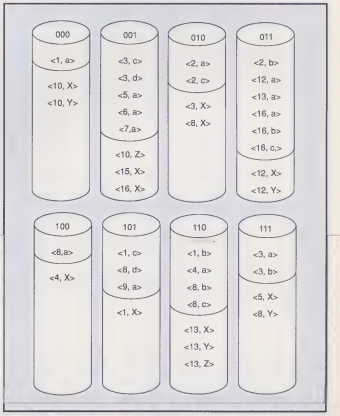


Figure 9. Initial data distribution.

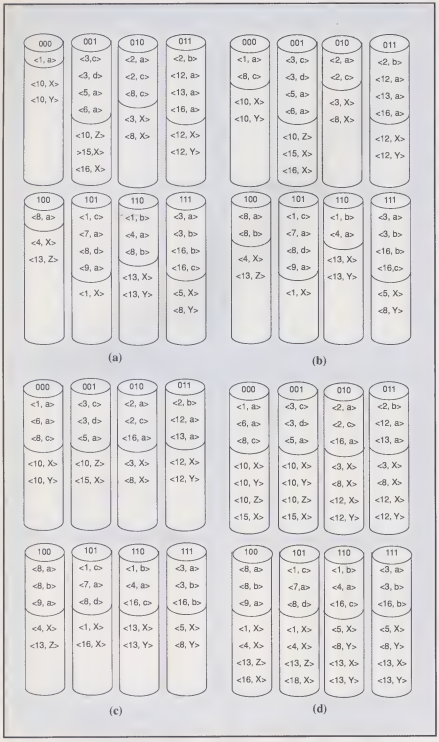


Figure 10. Broadcast-based Join (balancing: (a) step 1, (b) step 2, and (c) step 3). (d) represents the RCR stage.

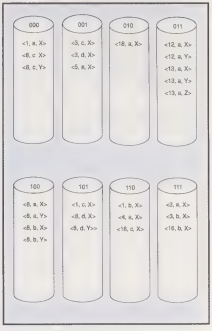


Figure 11. Final distribution of broadcast Join.

whose n–jth address bit is 0 keep S_1 and send S_2. Similarly, nodes whose n–jth address bit is 1 keep S_2 and send S_1. All nodes receive the sent tuples and merge them with those kept tuples. Sorted order is maintained. The GET_BUCKET(a, b) primitive places logical buckets a through b, inclusive, into send_set.

After n pairing steps, each node contains only tuples whose joining attributes fall within a disjoint subrange of the joining domain. Once both relations are redistributed, the local Join computation proceeds in parallel at each node.

Figure 12 shows a pseudocode description of the bucket data redistribution for a single relation. As with the balancing operation, both relations can be redistributed in parallel, with the one relation using link j at step j, while the second uses link $(j+1)$ mod n at step j. In this algorithm, however, either the portion of the algorithm that selects the buckets to exchange during each node-pairing step or the attribute-range-to-bucket-number assignment must be modified in the code that simultaneously repartitions the second relation. Also, in both the balancing and the bucket redistribution algorithms, simultaneous redistribution of both relations does not require separate links. If the two relations can be separated efficiently, both relations can be routed together on the same link. However, given independent communications processors and buffers, the use of multiple links generally reduces the data-transfer time. (Omiecinski and Tien describe a similar repartitioning algorithm that uses hashing to redistribute the relations.[10])

With the example in Figure 8, using a bucket Join algorithm that simultaneously redistributes both relations results in the following: In each of the three steps, a maximum of 5, 4, and 3 tuples, respectively, are routed in any internode communica-

tion. Note the significant difference between the maximal tuple transfer in the partitioning algorithm compared with the tuple-balancing algorithm. Figure 13 shows the result relation of the Join computation. As required, both Join approaches result in the same relation; however, the final distribution varies.

Characterizing performance potential

Here, I present experimental timings obtained from a portable hypercube-based database system. The system implementation issues are beyond the scope of this tutorial but can be found in an upcoming publication.[11] The evaluation focuses on only the two Join approaches presented here and includes a comparison of the timings obtained when the balancing step in the broadcast Join is omitted. Only timings for the Join operator are provided since the processing time of the Join greatly exceeds that of other common relational operators, making it the *computational* performance bottleneck in database processing.

I emphasize computational to differentiate between the I/O and CPU demands associated with database processing. The parallel I/O demands are commonly handled through parallelization of the Select operator, because the Select operators typically precede all other database operators in optimized query trees. No interprocessor communication exists in the Select operator, so performance measurements are not provided.

The timings presented were obtained using macros developed at Argonne National Laboratories[12] executing on an Encore Multimax running the Umax 4.2 operating system. These macros provided a fully portable, simulated, distributed-memory environment, allowing for system portability and independence from the hardware limitations of a particular vendor. The ability to vary hardware parameters comes at the expense of simulative overhead. Although the assumed hypercube architecture was simulated, all the database algorithms actually executed as though they were running on an actual hypercube. The Multimax configuration consisted of 16 processors, enabling each logical hypercube node to execute on a dedicated node in the Multimax. As the total system memory comprised 128 Mbytes, the entire database was memory resident.

```
begin    { BUCKET }

{ Redistributes data by attribute(s) values. Each bucket has a nonoverlapping
    subset of the global attribute domain. The union of the attribute value range
    of all the buckets is the global domain. Assumed is that all local tuples are
    already partitioned into local buckets and must only be routed to the
    appropriate processor.

    PREDEFINED FUNCTIONS:

        GET_BUCKET (k , j)      Function that moves all the tuples assigned
                                    to buckets k through j inclusive to
                                    a stated buffer (send_set)
        XOR (k, j)              Bitwise Exclusive OR of k and j
        rotate_right (k, j)     Rotate the binary representation of k, j bits
                                    to the right
        merge ( Y, buckets )    Mark the corresponding bucket for each
                                    tuple in buffer Y and move to the
                                    appropriate memory location

    INPUT:                      Local address of the node (X)

    PRIOR TO EXECUTION:         Relation randomly distributed across nodes

    EXECUTION TERMINATION:  Tuples distributed across nodes according
                                    to attribute value      }

    first := 0;
    last := N - 1;

    for j := 0 to n – 1 do begin

{ Select buckets to send      }
      if X[j] = 0 then begin

            send_set := GET_BUCKET ( (first + last + 1)/2, last );
            last := (first + last - 1)/2;
          end
          else begin

            send_set := GET_BUCKET ( first, (first + last - 1)/2 );
            first := (first + last + 1)/2;
          end
        end

      send ( XOR( X, rotate_right ( 1, j+1 ) ), send_set );
      receive ( XOR( X, rotate_right ( 1, j+1 ) ), receive_set );

{ Place received tuples into appropriate local bins      }
      merge ( receive_set, local_buckets );

    end { for }
  end { BUCKET }
```

Figure 12. Bucket-based partitioning pseudocode.

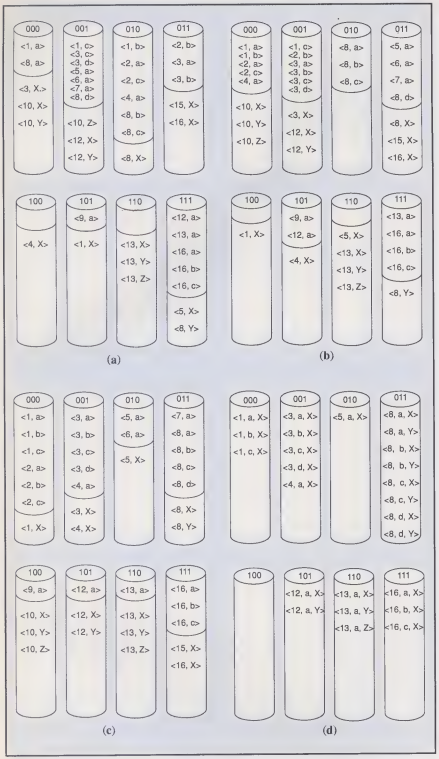


Figure 13. Bucket-based Join example: (a) step 1, (b) step 2, and (c) step 3. Part (d) represents the final result (bucket).

The emulative overhead significantly affects the actual timings observed in terms of absolute time. Hence, these timings should not be used for comparison against other systems, but only as a tool for comparing the two presented Join approaches. Because we are interested only in seeing the similarities and differences between these Join approaches, runs of a standard benchmark data set are not presented. Instead, a synthetic database that better illustrates the behavior of the presented approaches was constructed.

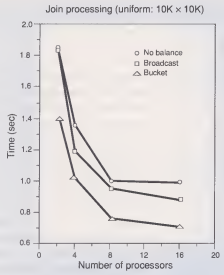
The database comprised relations of four different sizes: 1K, 10K, 100K, 1M tuples. Each tuple had 24 bytes. Relations were randomly generated using a uniform-attribute value distribution. The tuples of each relation were horizontally partitioned across the nodes using either a uniform (marked as "uniform" on the graph) or a bimodal skewed distribution (marked as "skewed" on the graph).

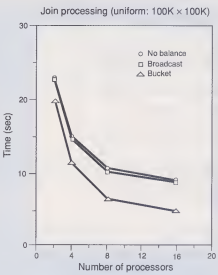
Results for multiple-size Joins are presented: 10K × 10K (uniform), 100K × 100K (uniform), 100K × 100K (skewed), 10K × 100K (uniform), 1K × 1M (uniform), and 1K × 1M (skewed). Each Join algorithm was run on four different hypercube configurations: two, four, eight, and 16 nodes. By definition, the smallest hypercube is of dimension 1, so single-node runs are not included in the timing presentation.

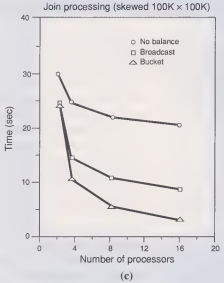
In general, the broadcast-based Join, as compared with a bucket-based Join, reduces the required communication at the expense of redundant computation. Redundant comparisons in the broadcast Join result from the unnecessary comparison of every tuple in the smaller relation, R_1, against all tuples in the larger relation, R_2. In the bucket-based Join, each tuple in R_2 is compared with only the relevant portion of R_1. In terms of communication requirements, unlike the bucket Join where both relations must be redistributed, the communication demands of the broadcast Join require that only R_1 be routed. Thus, when R_1 and R_2 are comparable in size, the bucket Join is better. However, if the difference in the communication times for the two Join approaches is significant (R_1 is much smaller than R_2), the broadcast Join is better. The precise size disparity is based on the degree of data skew and various system parameters.

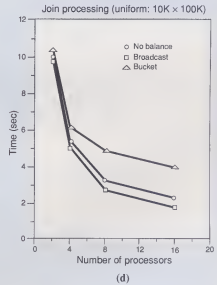
The timing results summarized in Figures 14a through 14f coincide with size disparity intuition. The scalability of the bucket Join exceeds that of the broadcast Join when the relations are comparable in size (Figures 14a-c), and the broadcast Join is superior when the relations differ in size (Figures 14c-f).

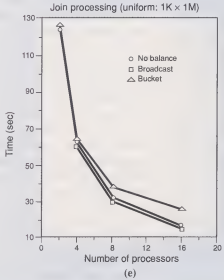
The graphs also show the performance obtained when the balancing step is not performed as part of the broadcast Join. When the tuples are uniformly distributed across the nodes, the overhead of balancing is negligible, but the reduction in processing time resulting from the balancing step is also negligible (Figures 14a, b,











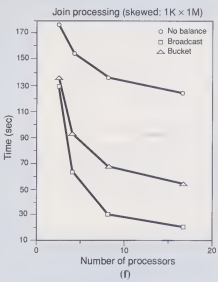


Figure 14. Experimental timing results.

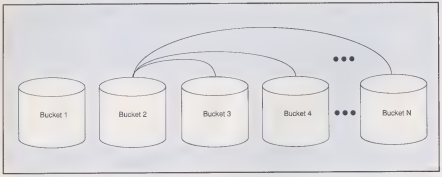


Figure 15. Theta-Join example.

```
begin   { Bucket-Based Less-Than Theta-Join }

   { Redistributes data of the smaller relation according to attribute(s) values
     for a LESS-THAN Theta-Join.

   PREDEFINED FUNCTIONS:
     XOR (k, j)        Bitwise Exclusive OR of k and j
     rotate_right (k, j)  Rotate the binary representation of k, j bits to the right

   INPUT:            Local address of the node (X)
                     An array indicating the lower bound of each bucket of
                       the larger relation (bounds)
                     An ordered set comprising all the tuples of the smaller
                       relation that are resident at the node (tuples).
                       tuples [i].attribute is the joining attribute value of
                       tuple i.

   PRIOR TO
   EXECUTION:        Larger relation distributed according to attribute values
                     Smaller relation randomly distributed

   EXECUTION
   TERMINATION:  Tuples distributed as required by Theta-Join  }

   first := 0;
   last := N - 1;
   for j := 0 to n - 1 do begin
     send_set := Ø ;
     num_tuples := | tuples |;
     if X[j] = 0 then begin
           for i := 1 to num_tuples do begin
                 send_set := send_set ∪ tuples [i];
                 if tuples [i].attribute ≥ bounds [(first + last + 1) / 2] then
                       tuples := tuples - tuples[i];
           end { for }
         last := (first + last - 1)/2;
       end { then }
       else begin
         for i := 1 to num_tuples do
               if tuples [i].attribute < bounds [(first + last + 1) / 2] then
                 send_set := send_set ∪ tuples [i];
         first := (first + last + 1)/2;
       end { else }

     send ( XOR( X, rotate_right ( 1, j+1 ) ), send_set );
     receive ( XOR( X, rotate_right ( 1, j+1 ) ), receive_set );
     tuples := tuples ∪ receive_set;
   end { for }
end { Bucket-Based Less-Than Theta-Join }
```

Figure 16. Theta-Join data-redistribution pseudocode.

d, and e). In fact, if the original tuple distribution is exactly even across the nodes, balancing will increase the total processing time. The observed minimal performance improvement results from the existence of some variance, albeit low, in the number of tuples resident at each node. However, as the variance of the tuple distribution increases (Figures 14b and f), the savings resulting from balancing become substantial. Thus, at least when faced with a skewed distribution of tuples across the nodes, the balancing step should be incorporated as part of the broadcast Join. All the results show that in a database system both Join algorithms should be implemented. The system should choose the approach to employ, depending on the input data set.

The Project operator

Similar to an aggregation operator, computing a Project operator (a multiscan, unirelational operator) involves two stages: local and global duplicate-tuple elimination. The first stage is local tuple elimination. Traditionally, local elimination requires each processor to sort its resident tuples and remove the duplicates. The second stage eliminates internode duplicates. As the communication requirements of the Project operator are a subset of the Join, either a broadcast or an attribute-partition scheme is possible.

If broadcasting is used, data replication is unnecessary because Project is a unirelational operator. Hence, instead of RCR, only relation compaction is required. That is, in each node pairing, the projected relation is compacted to only a single node, as in the aggregation algorithm. At each compaction, local duplicates are removed. Once relation compaction terminates, the cycling primitive is applied. In cycling, a copy of the compacted, projected relation is circulated. All nodes compare simultaneously their tuple set against the circulated tuples. If a duplicate is found, the local node number is compared with the node number generating the circulated tuple set. The local tuple is eliminated if the local node number is smaller.

A bucket-based Project implementation is nearly identical to a bucket-based Join algorithm. As in the Join, the domain is partitioned across the processors and the tuples are routed to the appropriate processor. To reduce the data routed, after each of the n steps, local duplicate elimination is performed. Thus, duplicates are eliminated as soon as possible.

Additional operators

Common set operators such as Union, Intersection, and Difference are typically incorporated into database systems. These multiscan operators can be implemented using the data-redistribution algorithms previously described. For example, the union of relations R[XYZ] and S[XYZ], written T[XYZ] = R[XYZ] ∪ S[XYZ], can be implemented by defining the new relation T[XYZ] as all the tuples of R and all the tuples of S, and then eliminating the duplicates in T, as in the Project operator.

The Intersection of R[XYZ] and S[XYZ], T[XYZ] = R[XYZ] ∩ S[XYZ], is the Join of the two relations in which only those tuples that match in all the attributes are kept, namely R.X = S.X, R.Y = S.X, and R.Z = S.Z. Finally, the difference of R and S, T[XYZ] = R[XYZ] – S[XYZ], is computed as a Join where only those tuples of R that *do not* join with any tuples in S are maintained.

Another popular operator that includes the Join described above as a special case is the *Theta-Join*. The Theta-Join of relations R[X] and S[Y], where $X \cap Y = \emptyset$, $A \in X$, $B \in Y$, and $\theta \in \{=, \neq, <, \leq, >, \geq\}$, written R[A θ B]S, is defined as $T[XY] = \{x \mid y \in R, z \in S, \text{such that } y[A]\ \theta\ z[B], y = x[X], \text{and } z = x[Y]\}$. Modifying a broadcast-based Join algorithm to encompass all the Theta-comparators requires only that the local equality comparison presently made at each node be replaced by the appropriate Theta-comparator.

Because a Theta-Join may require tuples in an attribute range to be compared against tuples in a set of attribute ranges, altering a bucket-based Join algorithm to support Theta-comparators requires not only that the local equality comparator be replaced by the appropriate Theta-comparator but that the actual tuple-redistribution algorithm be changed to support comparisons between an attribute range and multiple other attribute ranges.

Consider a Theta-Join where the comparator operator is Less Than. Relations R[ABC] and S[DE] are joined so R[B < D]S. Then, all the R tuples that map to bucket 2 based on the B attribute must be compared against all the S tuples that map to buckets 2 through N inclusive based on the D attribute, as illustrated in Figure 15.

The algorithm proceeds as follows. Initially, relation S is partitioned according to attribute values using the algorithm in Figure 12. Then, using the same bucket ranges, relation R is partitioned into buckets so all tuples in bucket i ($1 \leq i \leq N$) are routed to those processors that contain buckets j ($i \leq j \leq N$) of relation S. Figure 16 shows a pseudocode description of an algorithm that achieves the necessary bucket-based Theta-Join data routing. In contrast to the algorithm in Figure 12, where no additional copy of the data is generated, here multiple copies of the data assigned to the lower numbered buckets are produced, resulting in a vast volume of internode data transfer. Algorithms for the remaining Theta-Join comparators likewise result in a vast replication of data.

Operator implementation is crucial, but fully exploiting multiprocessors in database processing involves numerous additional database concerns not discussed in this tutorial. Such topics as data placement, execution-site selection, query optimization, security, and recovery must be addressed before large-scale (thousands of processors) multiprocessor database systems become an alternative to high-performance mainframe solutions for very large databases and complex database-based applications. ■

Acknowledgment

This work was partially performed while I was at Bellcore using the computational resources of the Syracuse University Northeast Parallel Architectures Center (NPAC), which is funded by DARPA/RADC contract #F306002-88-C-0031.

I wish to thank Paul Jackson, Mark Segal, Bruce Shriver, and the anonymous referees, whose input vastly improved this article.

References

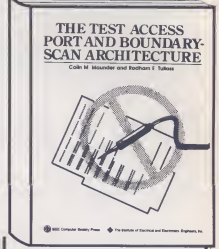
1. M. Stonebraker, "The Case for Shared Nothing," *Data Eng.*, Vol. 9, No. 1, Mar. 1986.

2. O. Frieder and C.K. Baru, "Query Scheduling and Site Selection Algorithms for a Cube-Connected Multicomputer System," *Proc. IEEE Eighth Int'l Conf. Distributed Computing Systems*, June 1988, CS Press, Los Alamitos, Calif., Order No. 865, pp. 94-101.

3. O. Frieder and G.E. Herman, "Protocol Verification Using Database Technology," *IEEE J. Selected Areas in Comm.*, Vol. 7, No. 3, Apr. 1989, pp. 324-334.

NEW RELEASE

from IEEE COMPUTER SOCIETY PRESS



The Test Access And Boundary-Scan Architecture

This tutorial discusses approaches to design-for-testability between computers and companies and explains its connection to IEEE Standard 1149.1. It begins by describing the circumstances that lead to the development of the standard, introduces boundary-scan techniques, and provides solutions to problems that are faced by this technology. Other key topics include: the structure of a typical board test program, testing and diagnosis of test logic, testing of boards, silicon implementation and related costs, interfacing to scan design, and applications to system debugging and emulation.

399 pages. September 1990. Hardbound. Illustrations. ISBN 0-8186-9070-4. Catalog No. 2070.

List $55.00 Member $44.00

To order your copy call--

1-800-CS-BOOKS

or in CA call--

714-821-8380

IEEE COMPUTER SOCIETY

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

4. C. Seitz, "The Cosmic Cube," *Comm. ACM*, Vol. 28, No. 1, Jan. 1985, pp. 22-33.

5. W.C. Athas and C.L. Seitz, "Multicomputers: Message-Passing Concurrent Computers," *Computer*, Vol. 21, No. 8, Aug. 1988, pp. 9-24.

6. D.J. DeWitt et al., "The Gamma Database Machine Project," *IEEE Trans. Knowledge and Data Eng.*, Vol. 2, No. 1, Mar. 1990, pp. 44-62.

7. D. Schneider and D. DeWitt, "A Performance Evaluation of Four Parallel Join Algorithms in a Shared-Nothing Multiprocessor Environment," *Proc. ACM SIGMOD Conf.*, ACM, New York, May 1989, pp. 110-121.

8. C.K. Baru, and O. Frieder, "Database Operations in a Cube-Connected Multicomputer System," *IEEE Trans. Computers*, Vol. 38, No. 6, June 1989, pp. 920-927.

9. C.K. Baru et al., "A Comparison of Join Algorithms for Hypercubes," *Proc. Fourth Hypercube Conf.*, SIAM, Philadelphia, 1989.

10. E. Omiecinski and E. Tien, "A Hash-Based Join Algorithm for a Cube-Connected Parallel Computer," *Information Processing Letters*, Vol. 30, No. 5, Mar. 1989, pp. 269-275.

11. O. Frieder, "A Database-Driven, Parallel Protocol Verification System Prototype," George Mason Univ., Dept. of Computer Science Tech. Report, TR-10-90, Oct. 1990. Submitted for publication.

12. E. Lusk et al., *Portable Programs for Parallel Processors*, Holt, Reinehart and Winston, Inc., 1987.

Further reading

The following list merely highlights some of the more recent parallel database efforts, and is not meant to be exhaustive. For additional background and earlier research, refer to the database machine textbooks listed in the last section below.

Hypercube-based algorithms

Arild, B., W. Baugsto, and J.F. Greipsland, "Parallel Sorting Methods for Large Data Volumes on a Hypercube Database Computer," *Proc. Sixth Int'l Workshop on Database Machines*, 1989.

Baru, C.K., et al., "Join on a Cube: Analysis, Simulation, and Implementation," in *Database Machines and Knowledge Base Machines*, M. Kitsuregawa and H. Tanaka, eds., Kluwer Academic Publishers, Boston, 1988, pp. 61-74.

Bratbergsengen, K., "Algebra Operations on a Parallel Computer: Performance Evaluation," in *Database Machines and Knowledge Base Machines*, M. Kitsuregawa and H. Tanaka, eds., Kluwer Academic Publishers, Boston, 1988, pp. 415-428.

Frieder, O., *Database Processing on a Cube-Connected Multicomputer*, doctoral dissertation, Univ. of Michigan, Ann Arbor, Mich., 1987.

Frieder, O., and P. Jackson, "On the Design, Implementation, and Evaluation of a Portable Parallel Database System," *Proc. IEEE Int'l Conf. Databases, Parallel Architectures, and Their Applications*, CS Press, Los Alamitos, Calif., Order No. 2035, Mar. 1990, pp. 516-518.

Frieder, O., "Fault Tolerance on a Hypercube: A Database Application," to be published in *J. of Systems and Software*, Nov. 1990.

Rishe, N., D. Tal, and Q. Li, "A Sequenced Hypercube Topology for a Massively Parallel Database Computer," *Proc. Second Symp. Frontiers of Massively Parallel Computation*, CS Press, Los Alamitos, Calif., Order No. 892, Oct. 1988, pp. 521-524.

Gamma (ring-based LAN)

DeWitt, D.J., et al., "Gamma — A High Performance Dataflow Database Machine," *Proc. Conf. Very Large Data Bases*, 1986.

DeWitt, D.J., et al., "A Single-User Evaluation of the Gamma Database Machine," in *Database Machines and Knowledge Base Machines*, M. Kitsuregawa and H. Tanaka, eds., Kluwer Academic Publishers, Boston, 1988, pp. 370-386.

Bubba (hypercube interconnection used for routing only)

Boral, H., "Parallelism in Bubba," *Proc. IEEE First Int'l Symp. on Databases in Parallel and Distributed Systems*, CS Press, Los Alamitos, Calif., Order No. 893, Dec. 1988, pp. 68-71.

Copeland, G., et al., "Data Placement in Bubba," *Proc. ACM SIGMOD Conf.*, ACM, New York, 1988, pp. 99-108.

Smith, M., et al., "An Experiment on Response Time Scalability in Bubba," *Proc. Sixth Int'l Workshop on Database Machines*, 1989.

Tandem (fault-tolerant bus-based) Architecture

Tandem Database Group, "Nonstop SQL, A Distributed, High-Performance, High-Availability Implementation of SQL," *Proc. Second Int'l Workshop High Performance Transaction Systems*, 1987.

Tandem Performance Group, "A Benchmark of Nonstop SQL on the Debit Credit Transaction," *Proc. ACM SIGMOD Conf.*, ACM, New York, 1988, pp. 337-341.

BBN Butterfly

Rosenau, T.J. and S. JaJodia, "Basic Database Operations on the Butterfly Parallel Processor: Experiment Results," Memorandum Report 6173, Naval Research Laboratory, 1988.

Oracle (database software for NCube's NCube-2 hypercube)

Oracle for the NCube 2 — Technology Overview, Doc. No. 50333-0789, Oracle, 1989.

Teradata DBC/1012 (a customized interconnection network)

DBC/1012 Data Base Computer System Manual, Doc. No. C10-0001-01, Teradata, 1985.

Database machine textbooks and tutorials

Boral, H., and P. Faudemay, *Database Machines*, Lecture Notes in Computer Science, Springer-Verlag, New York, 1989.

Hurson, A.R., L.L. Miller, and S.H. Pakzad, *Parallel Architectures for Database Systems*, CS Press, Los Alamitos, Calif., Order No. 838, 1989.

Kitsuregawa, M., and H. Tanaka, eds., *Database Machines and Knowledge Base Machines*, Kluwer Academic Publishers, Boston, 1988.

Ozkarahan, E., *Databases Machines and Database Management*, Prentice Hall, Englewood Cliffs, N.J., 1986.

Sood, A.K., and A.H. Qureshi, *Database Machines: Modern Trends and Applications*, Springer-Verlag, Heidelberg, Germany, 1986.

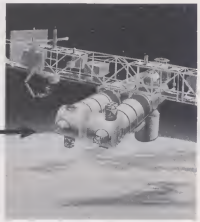
Su, S., *Database Computers: Principles, Architectures, and Techniques*, McGraw-Hill, New York, 1988.

Ophir Frieder is a faculty member of the Computer Science Department at George Mason University. From 1987 to 1990, he was a member of technical staff in the Applied Research Area of Bell Communications Research. His research interests include parallel and distributed architectures, database systems, operating systems, and medical imaging architectures.

Frieder received his BSc (1984) in computer and communications science and his MSc (1985) and PhD (1987) in computer science and engineering, all from the University of Michigan. He is a member of Phi Beta Kappa and the IEEE Computer Society.

Frieder can be reached at the Department of Computer Science, George Mason University, Fairfax, VA 22030-4444.

Alsys Ada cross-compilers get you there in no time.



It's time you knew that Alsys, the premier Ada company, offers a range of powerful and flexible cross-compilers for all microprocessors in the Motorola MC680X0 and Intel i80X86 families* to get your applications up and running fast.

Part of the reason for this speed are powerful development tools such as AdaProbe, a source level debugger and program viewer providing facilities to address both the execution properties of a program and its static structure. In addition, there's support for placing program components into ROM, and the Alsys Multi-Library Environment allowing program units to be shared among libraries for team programming.

With Alsys' full line of cross-compilers you'll discover impressive flexibility and power. There's a configurable run-time system giving you full control over tasks, interrupts and all components of your application. The debugger and transfer utility are configurable. Best of all, it's easy to take advantage of all this power because there are only a few files to modify.

When you need to get from here to there without getting lost somewhere in between, use a cross-compiler that knows the shortest route.

alsys
Ada Now!

*The 680X0 cross-compilers are hosted on VAX, SUN 3, Apollo, HP 9000/300, IBM PS/2, PC AT, and Compaq 386. The Intel cross-compiler is hosted on the IBM PC AT and Compaq 386.

In the US: Alsys, 67 South Bedford Street, Burlington, MA 01803-5152, Tel: (617) 270-0030.
In the UK: Alsys, Partridge House, Newtown Road, Henley-on-Thames, Oxon RG9 1EN Tel: 44(49)1579090
Elsewhere: Alsys, 29 Avenue de Versailles, 78170 La Celle St. Cloud, France Tel: 33(1) 3918.12.44

Check boxes that interest you. IEC11/90

- ☐ Please send me a Technical Summary on the Motorola MC680X0 cross-compilers.
- ☐ Please send me a Technical Summary on the Intel i80X86 cross-compilers.
- ☐ Please send me a copy of your "Many Facets of Quality" brochure.

Name _____

Address _____

City _____ State _____ Zip _____

Phone ( )

Mail to: Alsys, Inc., 67 South Bedford Street, Burlington, MA 01803-5152

Reader Service Number 7

A Functional Correctness Model of Program Verification

Marvin V. Zelkowitz
University of Maryland

Translating a problem description into a computer solution is central to programming, but the process is generally ill-defined, error-prone, and expensive. Some researchers are using formal mechanisms to describe a problem and algorithmic processes to convert the problem statement into a program. The general approach is to describe a problem in a specification language with well-defined syntax and semantics. This reduces the problem to developing a source program that meets the specification. Among the specification languages investigated, axioms[1] and algebraic models[2] predominate.

Verification is just one of many approaches to producing quality software. (The sidebar on the following page shows where it fits in the overall scheme.) The functional approach described in this article is one alternative, but it is not meant as a panacea for poor requirements and specifications analysis. Regardless of the model employed, verification is a precise, formal, and difficult undertaking. Some applications do, however, lend themselves to a functional approach that has not been adequately described in the literature.

The University of Maryland uses a functional correctness model as part of its introductory computer science course. The model was originally developed by Mills,[3] who, with others,[4-6] has since refined it. The idea is to express a specification as a mathematical function, develop a program, and prove that the function implemented by that program is the same as the specification function. The system used at the university meets several of Dijkstra's criteria[7] for the teaching of formalism to express programs.

This model's verification conditions depend only on elementary symbolic execution of a trace table. An easy-to-learn technique, it's used in a freshman computer science course.

In this article, the method is applied to rather simple programs. However, even in large complex implementations, the techniques can be applied informally to determine the functionality of complex interactions.

Functional model of a program

Specifications. A specification is a mathematical description of a problem to be solved. Let α be a string representing a source program. For example, a Pascal program is just the linear string

program main(input, output); ... end.

We express the mathematical function denoted by program α by a *box notation.** $[\alpha]$ represents the function that com-

*$[p]$ is often written as $\overline{p}$ in other papers on the subject.

0018-9162/90/1100-0030$01.00 © 1990 IEEE

Approaches to producing quality programs

Testing. In the oldest technique, testing, programs are executed using sample data that is representative of the data processed under actual use. If the data is chosen appropriately, most errors can be found. But, as Dijkstra has observed, testing can only show the presence of bugs, not their absence. In most large implementations, testing is the most feasible and generally the only usable technique.[1]

Design methodology. Good techniques produce well-structured programs, which minimize faulty logic and hence errors. Techniques like structured programming, data abstractions, top-down design, and object-oriented programming help the programmer think more clearly about the programming process. While these methods are great aids in producing quality programs, the programs must still be checked using other techniques for the eventual programming glitch.[2]

Verification. Under this technique, programs are viewed as formal objects developed from a set of precise specifications. Once developed, they are guaranteed to produce the output given in the specifications. This is the model developed in this article. Although it guarantees the stated output, the technique has problems: there is no guarantee that the specifications are correct, and the development of proofs is extremely difficult. The major techniques are *axiomatic*, where programs are considered extensions to the predicate calculus[3]; *algebraic*, which views programs as equations[4]; and *functional*, the approach in this tutorial.

Reliability. Reliability is the probability that software will not cause a system failure for a specified period of time, whether or not the data presented to the software meet the program's specifications. This contrasts to the above definition of verification. Correct programs may be very unreliable, and reliable programs may not be correct. For example, consider two watches — one stopped and the other two hours late. The stopped watch is correct twice a day; the late watch is never correct. However, the stopped watch is highly unreliable, while the late one is quite reliable.

Testing is often the best method to show good reliability.[5] *Software safety* is a related topic that addresses reliability and the probability of embedded systems causing physical harm to individuals.[6]

As can be seen by the above list, all of the techniques are useful, but they are difficult to use effectively. Improving these methods is a major focus of software engineering research.

References

1. R. DeMillo et al., *Software Testing and Evaluation*, Benjamin Cummings, Menlo Park, Calif., 1987.
2. E. Yourdon, *Writings of the Revolution: Selected Readings on Software Engineering*, Yourdon Press, New York, 1982.
3. D. Gries, *The Science of Programming*, Springer-Verlag, New York, 1981.
4. J. Guttag, "Notes on Type Abstraction (Version 2)," *IEEE Trans. Software Eng.*, Vol. 6, No. 1, Jan. 1980, pp. 13-23.
5. D.L. Parnas, J. van Schowen, and S.P. Kwan, "Evaluation of Safety-Critical Software," *Comm. ACM*, Vol. 33, No. 6, June 1990, pp. 636-648.
6. N. Leveson, "Software Safety: Why, What and How," *Computing Surveys*, Vol. 18, No. 2, 1986, pp. 125-163.

putes the same values as program α.

Although a function is the intuitive model of a specification, often we simply want one feasible solution out of many possibilities. In choosing one optimal strategy from several equivalent ones (for example, equivalent optimal moves in a game-playing program), we usually do not care which solution the program employs. Because of this, we only need to define a specification as a *relation*. If r is such a specification relation, it is equivalent to a program p by the following *correctness theorem*[5]:

> Program p is correct with respect to specification relation r if and only if domain($r \cap$ [p]) = domain(r).

In other words, if we take the subset of r from those pairs in r that are also in function [p] (that is, $r \cap$ [p]), we have a function. If this function has the same domain as r, then [p] includes a pair of values for each member of relation r, and we get a feasible (or correct) implementation of the specification. In what follows, however, we use the simpler case. We have chosen the more restricted specification function f, instead of the more general relation r, with the corresponding correctness theorem of $f \subseteq$ [p].

Programs. A program is a sequence of declarations followed by a sequence of statements. Each maps a set of values for every variable in the program into a new set of values. Using denotational semantics, we can define the meaning of such a program as follows:

If *var* is a set of variable names and *val* is a set of values, a *state* is a function with the signature *state* : *var* → *val*. A state represents the formal model for program storage (for example, activation records).

If *expr* is an expression, [expr] is a function that maps a state into values, or [expr] : *state* → *val*. For example, if (x,a) and (y,b) represent entries in the state function S representing variables x and y, then [x+y] (S) is defined to be the function with [x] (S) + [y] (S) as a value. If we define [x] (S) to be $S(x)$, then $S(x) = a$, which agrees with our intuitive definition that [x+y] $(S) = a+b$.

If s is a Pascal statement, then [s] is a function that maps a state into a state — that is, each statement maps a set of values for all variables into a new set of values. If s is a declaration, then the resulting state includes a (var, val) pair for the newly declared variable. For example, if s is the state $\{(x,1),(y,2)\}$, then the function [y:=x] applied to s results in the state $\{(x,1),(y,1)\}$.

It is easy to see the correspondence between sequential execution and function composition. If s is a sequence $s_1, s_2, \ldots, s_n$ of statements, then $[s] = [s_1, s_2, \ldots, s_n] = [s_1] \circ [s_2] \circ \ldots \circ [s_n] = [s_n] (\ldots ([s_2] ([s_1])) \ldots)$.

The function [p] for "program main (input, output); begin s1; s2; ... end." is given by [program main (input, output)] $\circ$ [s_1] $\circ$ [s_2] $\circ \ldots \circ$ [.] where the signature for [program...] is *val* → *state*, for [.] is *state* → *val*, and *state* → *state* for all other statements. Hence, a program maps a value to a value and is composed of functions that map states to states. (Details of how to handle individual statement types like assignments,

conditionals, and iteration are given later.)

Developing a program requires several separate activities:

(1) designing a specification that expresses the task to be performed,
(2) refining that specification into a formal explicit statement that captures the specification's intended functionality, and
(3) developing a program that correctly implements that functionality.

Most of this article concerns the transition between the last two steps. Techniques will be given that aid in this transition and help show that both formalisms have equivalent functionality.

Applications. With this notation, three separate activities — verification, program design, and reverse engineering — can be investigated:

(1) If f is a function and if p is a program, show $[p] = f$ — that is, verification.

(2) If f is a function, develop program p such that $[p] = f$ — that is, program design. As a practical matter, we only care that $f \subseteq [p]$, since any value in $[p]$ and not in f represents a value computed by the program that is outside its specifications and not of interest to us.

(3) If p is a program, then find a function f such that $[p] = f$ — that is, reverse engineering. Given a program, determine its specifications. Some heuristics are given, but the basic method is to "guess" a solution and show by methods 1 and 2 above that it is the correct solution.

Symbolic execution

Symbolic execution is an aid in showing functional composition. To show that $[p] = f$, we symbolically execute program p and show that the resulting function is the same as f.

For example, consider the Pascal sequence

```
x:= x+1;
y:= x+y;
x:= y+1
```

Since we know that

$$[x:=x+1;\ y:=x+y;\ x:=y+1] = [x:=x+1] \circ [y:=x+y] \circ [x:=y+1]$$

we can symbolically execute each statement function. We use a trace table where we write, under "Part," the relevant statement function and, under each relevant variable, the new value that results from that execution. In the statement function, we substitute the value of each variable at that point in the computation. This results in a new function that can transform each variable into its new value.

For the above Pascal sequence, we get the following trace table:

Part	x	y
x:= x+1	x+1	
y:= x+y		(x+1)+y
x:= y+1	(x+1+y)+1 = x+y+2	

This states that simultaneously x is transformed by the function [x:=x+y+2] and y is transformed by [y:=x+y+1].

The extension of the trace table to handle conditionals (for example, if statements) requires a condition column. We write the predicate that must be true at that point for that execution path to proceed, and we develop trace tables for each path through the program.

For example, the program sequence

```
x:= x+y;
if x>y then
    x:= x–1
```

has two possible execution sequences, ($x>y$ and $x\leq y$) and two corresponding traces:

Part	Condition	x	y
x:= x+y		x+y	
if x>y	(x+y)>y		
x:=x–1		(x+y)–1	

and

Part	Cond.	x	y
x:= x+y		x+y	
if x>y	(x+y)≤y		

These two tables represent the following: if $x+y>y$, the function is [x:= x+y–1]; if $x+y\leq y$, the function is [x:= x+y]. The next section shows how to write this as a conditional assignment function.

Design rules

As stated earlier, software development consists of (1) designing the specification, (2) formalizing the specification, and (3) developing the source program. We use a functional notation for step 2 that is closely tied to the eventual Pascal source program. This notation includes (1) concurrent assignment, (2) conditional assignment, and (3) loop verification. This notation was strongly influenced by McCarthy's work on Lisp.

Designing assignment statements. Concurrent assignment is defined as simultaneous assignment. The function

$$(x,y,z := y,z,x)$$

simultaneously accesses the current values of variables y, z, and x and stores them, respectively, into variables x, y, and z. Mathematically, the state function that results will have the same values for all state variables other than x, y, and z, and those three will have new values.

Given statement p, showing that $[p]$ does implement this concurrent assignment is simply a matter of building its trace table. The more interesting problem is how to develop p, given some concurrent assignment as its specification. This leads to three design heuristics for concurrent assignment:

(1) All values on the right side of the intended concurrent assignment (that is, all values needed by a left-side variable) must be computable at each step.

(2) At each step, if a variable can be assigned its intended value, do so. Otherwise, introduce a temporary variable, and assign it a value that must be preserved.

(3) Stop when all variables on the left side of the intended concurrent assignment have been assigned their intended values (that is, when finished).

If we "execute" a trace table as we develop each Pascal assignment statement, we are also verifying that the design works as we wish. Once the values in the trace table are the desired values, we have shown that the assignment statements written do indeed implement the intended concurrent assignment.

Remember, however, that the three design rules are heuristics, not an algorithm. They indicate how to search for a solution and how to check if the solution is correct, but they do not give the solution. We have not replaced the art of programming by an implementable methodology that automatically builds correct programs from specifications.

Designing conditional statements. The conditional assignment is the formal model of conditionals. If b_i is a Boolean condition and c_i is a design function, then a conditional statement has the syntax

$$(b_1 \rightarrow c_1) \mid (b_2 \rightarrow c_2) \mid \ldots \mid (b_n \rightarrow c_n)$$

with the semantics of evaluating each b_i in turn, and setting the value of the conditional to be c_i for the first b_i that is true. If all b_i are false, then the statement is undefined. (This is similar to the *cond* of Lisp.) If b_n is the default case (that is, the expression *true*), then it can be omitted, with the last term becoming (c_n). The *identity* function is written as ().

We'll use several theorems involving conditional statements in this article. They can be verified by simple trace tables:

(1) Conditional $(a \rightarrow b) \mid (\text{not}(a) \rightarrow c)$ has the same meaning as $(a \rightarrow b) \mid (c)$.

(2) Conditional $(a \rightarrow (b \rightarrow c)) \mid (b \rightarrow c)$ has the same meaning as $(a$ and $b \rightarrow c)$.

(3) Conditional $(a \rightarrow c) \mid (b \rightarrow c)$ has the same meaning as $(a$ or $b \rightarrow c)$.

(4) Conditional $(a \rightarrow (b$ or $c)) \mid (a \rightarrow c)$ has the same meaning as $(a \rightarrow b) \mid (a \rightarrow c)$.

The Pascal source program for this design is simply a series of if statements that test each condition in turn. For example, given

$$(b_1 \rightarrow c_1) \mid (b_2 \rightarrow c_2) \mid \ldots \mid (b_n \rightarrow c_n)$$

the Pascal program can be written directly as

```
if b1 then c1
else if b2 then c2
else if b3 then c3
...
```

If all the b_i are false, since the Pascal code is everywhere defined, the specifications are actually a (correct) subset of this source program.

Verifying assignment and conditional statements. Assume p is the program to be verified and it consists of only if and assignment statements. There are only a finite number of execution paths through the program. For each path, compute the condition that must be true to execute that path, and use a trace table to determine what happens to the variables by executing that given path. Assume p_1, p_2, ... are the conjunctions of all conditions on each execution path, and a_1, a_2, ... are the corresponding concurrent assignments. The function f that this implements is

$$(p_1 \rightarrow a_1) \mid (p_2 \rightarrow a_2) \mid \ldots \mid (p_n \rightarrow a_n)$$

Consider the following example:

```
x:=x+y;                 (* 1 *)
y:=y–x;                 (* 2 *)
if x+y>0 then
        y:=x+y   (* 3 *)
    else
        y:=–x–y  (* 4 *)
```

This has two execution sequences, 1-2-3 and 1-2-4, with two different traces.

(1) If is true:

Part	Cond.	x	y
x:=x+y		x+y	y
y:=y–x		x+y	–x
if x+y>0	(x+y)–x>0		
y:=x+y			x+y–x=y

(2) If is false, so not(if) is true:

Part	Cond.	x	y
x:=x+y		x+y	y
y:=y–x		x+y	–x
if x+y>0	(x+y)–x≤0		
y:=–x–y			–(–x)
			–(x+y)
			=–y

This gives the function:

$$(y>0 \rightarrow x,y := x+y,y) \mid (y \leq 0 \rightarrow x,y := x+y,-y)$$

Or, since the assignment to y (that is, $(y>0 \rightarrow y := y)$ and $(y \leq 0 \rightarrow y := -y)$) is just function abs($y$), the function reduces to

$$(x,y := x+y,\ \text{abs}(y))$$

We could have left our answer as a conditional assignment, but replacing it as a concurrent assignment using the absolute value function leads to a more understandable solution. Knowing when (and how) to apply such reductions is probably as complex an issue as any encountered in axiomatic verification.

Verifying while loops. To handle full program functionality, we must address loops. Given a functional description f and a while statement p, we first describe three verification rules that prove that f and p are equivalent. These will be denoted V.I-III. Once we have these verification conditions, we would like to use them as design guidelines to help develop p, given only f. We call these five design rules V.I-V.

The while statement [while b do d] is defined recursively via the if statement to mean[3]

```
[while b do d] = [if b then begin d;
                  while b do d end]
```

That is, if b is true, perform d and repeat the while statement. Via a simple trace table we get the same result as

```
(**) [while b do d] = [if b then d;
                       while b do d] ∘ [while b
                       do d]
```

Let f be the meaning of the while statement, that is, f = [while b do d]. By substituting back into (**) above, we get the first condition that

(V.I) f = [if b then d] ∘ f

What other conditions on f ensure it is indeed the specification of the while statement? If f is undefined for some input x, then both sides of the equation are undefined. To ensure that this cannot happen, we require that f be defined whenever [while] is defined, or that domain([while]) ⊂ domain(f). (Note: For ease in reading, we will use [while] to stand for [while b do d]).

Similarly, if [while] is everywhere the identity function, then any f will fulfill the equation since the recursive equation reduces to $f = () \circ f = f$. Thus, we must also have domain(f) ⊂ domain([while]). This yields

(V.II) domain(f) = domain([while b do d])

Consider any state $s \in$ domain([while]). If [b] (s) is true, that is, expression b in state s is true, then from (**), s_1 = [d] (s) and $s_1 \in$ domain([while]). This will be true, for s_2, s_3, and so on, until at some point [b] (s_n) is false and both [if b then d] (s_n) and [while b do d] (s_n) equal s_n.

This s_n is a member of domain([while]) and of range([while]). More importantly, if [b] (s) evaluates to false, then [while] (s) = s. Or, stated another way, [while] (s) = s for all states s where [b] (s) is false. This is just a restriction on the [while] function to those states where b is false, which is the function (not(b) → [while]). This must be

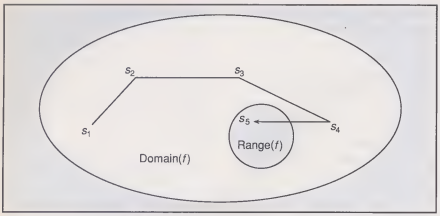


Figure 1. Domain and range of the while function.

equal to the identity function (), also restricted to the same domain, or just (not(*b*) → ()). Any candidate function *f* must also have this property, yielding the third constraint

(V.III) (not(b) → f) = (not(b) → ())

Designing while loops. Consider the problem of designing a loop. Given a specification *f*, how can we design a while from the three statement verification conditions given above?

From V.III, the while terminates when [b] evaluates to false, and range([while]) is just the set of states where [b] is false. But since we can apply [while] to this state initially, it is also part of domain([while]).

Therefore, as shown in Figure 1, range([while]) ⊂ domain([while]). Since *f* must also have this property, we get the first design constraint:

(D.I) range(f) ⊂ domain(f).

Similarly, we have shown that for an *s* where [b](*s*) is false, [while](*s*) = *s*, we must also have *f*(*s*) = *s*, because if [b] (*s*) is false, the body *d* is not executed. But these are just the points in range(*f*). Therefore, we get the second design constraint:

(D.II) if s ∈ range(f), then f(s) = s.

D.I and D.II must be true if *f* is the meaning of a while statement. Therefore, they show the existence of a possible solution.

From D.II, we know *f* must be an identity on range(*f*) in order to be implemented with a while. We can restate this as:

(D.III) [b] evaluates to true in domain(f)-range(f) and false in range(f).

Verification example using the functional correctness model

This example shows that the function $f = (A \leq B \rightarrow A,B := (B-(B-A)/2), (B-(B-A)/2) \mid ( ))$ is implemented by the source program

```
1    while A<B do
2      begin
3      A := A+1;
4      if A<B then
5        B := B–1
6      end
```

where *A* and *B* are integers and division means integer truncated division (for example, 1/2 = 0).

The approach we follow is to first determine the functionality of the assignment statement (line 3), then the if statement (lines 4-5), then the entire begin block (lines 2-6), and finally the functionality of the entire segment (lines 1-6).

Line 3. A := A+1 is just the concurrent assignment (*A* := *A*+1).

Lines 4-5. d_1 = [if A<B then B := B–1]

If *A*<*B* is true, evaluate the function *B* := *B*–1, and if it is false, skip the then statement and do nothing, for example, the identity function. The conditional assignment can be written as

$d_1 = (A<B \rightarrow B := B-1) \mid ()$

Lines 2-6. d_2 = [begin A := A+1; if A<B then B := B–1 end]

d_2 = [A:=A+1] ∘ [if A<B then B:=B–1] = (*A*:=*A*+1) ∘ d_1
= (*A*:=*A*+1) ∘ ((*A*<*B* → *B*:=*B*–1) | ())

Develop a trace table for the begin block. There will be two paths through this block (for example, first and second alternatives for d_1). Hence, there will be two trace tables:

Part	Cond.	A	B
3: A:=A+1		A+1	
4: if A<B	(A+1)<B		
5: B:=B–1			B–1

and

Part	Cond.	A	B
3: A:=A+1		A+1	
4: if A<B	(A+1)≥B		

Similarly to the assignment design, we develop the while loop via

(D.IV) Develop d so that all values are preserved for f.

(D.V) Show that f is everywhere defined, that is, the loop must terminate for all $x \in$ domain(f).

Given function f, we develop a while statement such that [while] = f as follows:

(1) *Existence.* Verify conditions D.I and D.II. If these cannot be satisfied, then no such while statement can be written.

(2) *Range determination.* Use D.III to develop some predicate b such that [b] is false on range(f) and true on domain(f)-range(f). Since f and [while] are to be the same function, b becomes the predicate for the loop.

(3) *Loop body.* Use D.IV to develop an appropriate d. These guidelines do not give absolute solutions to this problem, but they do indicate how to verify whether d, once found, is a solution. It is comparable to finding the loop invariant in an axiomatic proof.

(4) *Termination.* Prove that the selected b and d cause the loop to terminate (condition D.V). If proven, since step 2 shows that [b](x) is false for $x \in$ range(f), this shows that the loop will terminate with some x in this range.

Examples. For two simple examples of this method, see the sidebars below and on page 38. The first example verifies a program with its specifications; the second example shows the design of a program from its functional specification. For a more complex example, see Gannon, Hamlet, and Mills.[4]

Data abstraction and representation functions

The discussion so far has concentrated on the process of developing a correct procedure from a formal specification. However, program design also requires appropriate handling of data.

Data abstractions. A data abstraction is a class of objects and a set of operators that access and modify objects in that class. Such objects are usually defined via the *type* mechanism of a given programming language, and a module is created consisting of such a type definition and its associated procedures.

Isolation of the type definition and invocation of the procedures that operate on such objects are crucial to the data abstraction model. Each procedure has a well-defined input/output definition. The implementor is free to modify any procedure within a module as long as its input/output functional behavior is preserved, and any use of such a procedure can only assume its functional specification. The result is that, rather than viewing a program as a complex interaction among many objects and procedures, a program can be viewed as the interaction among a small set of data abstractions — each relatively small and well defined.

We then get

$$d_2 = (A+1<B \rightarrow A,B := A+1,B-1) \mid (A := A+1)$$

Lines 1-6. Show f = [while A<B do begin A:=A+1; if A<B then B:=B–1 end]

We must show that function f meets the three verification rules. We will do this in the order V.II, V.III, and V.I.

(1) *Show V.III.* (not($A<B$) $\rightarrow f$) = (not($A<B$) $\rightarrow$ ( ))

(not($A<B$) $\rightarrow f$) =

($A \geq B \rightarrow (A \leq B \rightarrow A,B := B-(B-A)/2, B-(B-A)/2) \mid ( ))$ =

($A \geq B$ and $A \leq B \rightarrow (A,B := B-(B-A)/2, B-(B-A)/2)) \mid (A \geq B \rightarrow ( ))$ =

($A=B \rightarrow A,B := B-(B-A)/2, B-(B-A)/2) \mid (A \geq B \rightarrow ( ))$ =

($A=B \rightarrow A,B := A,B) \mid (A \geq B \rightarrow ( ))$ =

($A \geq B \rightarrow ( )$)

(2) *Show V.II.* domain(f) = domain([while])

f is defined for all A and B. For $A \leq B$, an explicit assignment is given, and for all other A and B, f is the identity function.

The [while] function is also defined for all A and B. If $A \geq B$, the body of the while does not execute giving the identity function for such A and B. If $A<B$, then for each pass through the loop, A is increased by 1 and B may be decremented by 1. At some point, $B-A$ must reach 0 or become negative. If $B-A \leq 0$, then $B \leq A$ and the while loop terminates. So for all A and B, the while statement must terminate and will generate some value for A and B.

(3) *Show V.I.* f = [if b then d] $\circ f$

The meaning of the body of the if statement (d) is the previously defined function:

$$d_2 = (A+1<B \rightarrow A,B := A+1,B-1) \mid (A := A+1)$$

The problem then reduces to showing that

$$f = [\text{if A<B then } (A+1<B \rightarrow A,B:=A+1,B-1) \mid (A:=A+1)] \circ ((A \leq B \rightarrow A,B := B-(B-A)/2, B-(B-A)/2) \mid ( ))$$

We will generate the set of functions that represent each separate path through each possible trace table. If we let c_1 be the if expression $A<B$, c_2 be $A+1<B$ in d_2, and c_3 be $A \leq B$ in f, then there are six possible paths through this function yielding six different trace tables, each deriving a different function g_i:

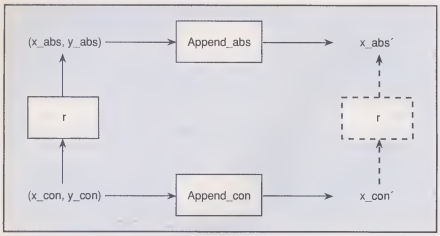


Figure 2. Commuting representation diagram.

Languages such as Ada (or C++) allow data abstractions to be built relatively easily, since the object type can be specified as the *private* part of a *package* (or *class*) specification. Only the body of the package has access to the type structure, while other modules have access only to the procedure names contained in the module.

However, even in older languages, such as C or Pascal, data abstractions form a good model of program design. Although not automatically supported by the language, with reasonable care, programs can be designed that adhere to the data abstraction guidelines.

Representation functions. A procedure within a data abstraction translates a high-level description of a process into a lower level programming language implementation. For example, suppose character strings up to some predefined maximum value are needed. Pascal only defines fixed-length strings; therefore, we must implement this as objects using primitive Pascal data types.

In procedures outside the defining module, we would like to refer to these objects (for example, call them *Vstrings*) and be able to operate on them. While inside the module, we need to operate on their Pascal representation (arrays of characters). In the former case, we call such functions *abstract functions* that define the functional behavior of the operation, while we call the latter *concrete functions* that give the implementation details.

For the Vstring example, we could define the string via an abstract comment containing the functional definition:

c_1	c_2	c_3	Function
true	true	true	g_1
true	false	true	g_2
true	true	false	g_3
true	false	false	g_4
false	—	true	g_5
false	—	false	g_6

(Note: If c_1 is false, then d_2 is not evaluated, giving only six possibilities rather than the full complement of eight that normally occurs with three predicates.)

We need to show that

$f = g_1 \mid g_2 \mid g_3 \mid g_4 \mid g_5 \mid g_6$

For g_1:

Part	Cond.	A	B
c_1 is true	A<B		
c_2 is true	A+1<B	A+1	B−1
c_3 is true	A+1≤B-1	B−1−(B−1−(A+1))/2= B−(B−A)/2	B−1−(B−1−(A+1))/2= B−(B−A)/2

The resulting predicate, $(A<B)$ and $(A+1<B)$ and $(A+1\leq B-1)$, reduces to $A<B-1$; $g_1 = (A<B-1 \rightarrow A,B := B-(B-A)/2, B-(B-A)/2)$

For g_2:

Part	Cond.	A	B
c_1 is true	A<B		
c_2 is false	A+1≥B	A+1	
c_3 is true	A+1≤B	B−(B−(A+1))/2	B−(B−(A+1))/2

The resulting predicate is $(A<B)$ and $(A+1\geq B)$ and $(A+1\leq B)$, which reduces to $A=B-1$.

By substituting $B-1$ for A, we get

$$g_2 = (A=B-1 \rightarrow A,B := B-(B-(A+1))/2, B-(B-(A+1))/2)$$

$$= (A=B-1 \rightarrow A,B := B-(B-(B-1+1))/2, B-(B-(B-1+1))/2)$$

$$= (A=B-1 \rightarrow A,B := B,B)$$

However, if $A=B-1$, then $(B-A)/2=0$. Thus we can write g_2 as

$$g_2 = (A=B-1 \rightarrow A,B := B-(B-A)/2, B-(B-A)/2)$$

For g_3:

Part	Cond.	A	B
c_1 is true	A<B		
c_2 is true	A+1<B	A+1	B−1
c_3 is false	A+1>B−1		

{abs: x_abs = <x_1, x_2, ..., x_n>}

The concrete representation of a Vstring could be

{con: x_con: record
chars: array [1...maxval] of char;
size: 0..maxval
end;}

To show that both representations are the same, we define a *representation function* that maps concrete objects into abstract objects. It does this by mapping a state into a similar state, leaving all data unchanged except for those specific objects. Let r map a concrete object into its abstract representation. If Cstrings is the set of concrete strings (that is, the set of variables defined by the above record description) and Vstrings is the set of abstract strings, then we define a representation function r with the signature

r: state → state

such that $r = \{(u,v) \mid u=v$, except that if $u(x) \in$ Cstrings, then $v(x) \in$ Vstrings$\}$. This simply means that u and v represent the same set of variables in the program store, except that each occurrence of a concrete variable in u is replaced by its abstract definition in v.

For each implementation of a string, we have its abstract meaning given by function r:

x_abs:=<x_con.chars[i] | $1 \le i \le$ x_con.size>

The purpose of a procedure in an abstraction module is to implement an abstract function on this abstract data. For example, if we would like to implement an Append operation, we can define x:=Append (x,y) as

{abs: x_1,..., x_n,..., x_{n+m} := x_1,..., x_n, y_1,..., y_m}

Similarly, we can define a concrete implementation of this same function as

{con: x.chars[n+1],...,
x.chars[n+y.size],
x.size := y.chars[1],..., y.chars[y.size],
x.size+y.size}

If x_con and y_con represent the concrete implementations of Vstrings x and y, and if x_abs and y_abs represent their abstract representation, and if Append_con and Append_abs represent the concrete and abstract functions, we have

x_con′ := Append_con (x_con, y_con)
x_abs′ := Append_abs (x_abs, y_abs)

We want to know if both the concrete and abstract functions achieve the same result, or if the abstract representation of what we get by implementing Append_con is the same as our abstract definition of Append. This is just the result: Is r(x_con′)=x_abs′? We say that the representation diagram of Figure 2 *commutes* (that is, either path from (x_con, y_con) to x_abs′ gives the same result). We have to show that r applied to x_con′ gives us x_abs (for example, x_1, x_2,..., x_n, y_1,..., y_m).

As given by our earlier correctness the-

This leads to the condition (A<B) and (A+1<B) and (A+1)>(B−1). We get (A<B−1) and (A>B−2), which is the null function.

For g_4:

Part	Cond.	A	B
c_1 is true	A<B		
c_2 is false	A+1≥B	A+1	
c_3 is false	A+1>B		

The resulting condition is (A<B) and (A+1≥B) and (A+1>B). But (A<B) and (A+1>B) are mutually disjoint, making g_4 null.

For g_5:

Part	Cond.	A	B
c_1 is false	A≥B		
c_3 is true	A≤B	B−(B−A)/2	B−(B−A)/2

The resulting condition is (A≥B) and (A≤B) or A=B. For A=B, B−(B−A)/2 = B = A; g_5 = (A=B → A, B := A,B).

For g_6:

Part	Cond.	A	B
c_1 is false	A≥B		
c_3 is false	A>B	A	B

The resulting condition is (A≥B) and (A>B) or just (A>B); g_6 = (A>B → A,B := A,B).

Next, show $f = g_1 \mid g_2 \mid g_3 \mid g_4 \mid g_5 \mid g_6$. In this example, since g_3 and g_4 are null, we have to show that $f = g_1 \mid g_2 \mid g_5 \mid g_6$.

$g1 \mid g2 \mid g5 \mid g6 =$
(A<B−1 → A,B := B−(B−A)/2, B−(B−A)/2) |
(A=B−1 → A,B := B−(B−A)/2, B−(B−A)/2) |
(A=B → A,B := A,B) |
(A>B → A,B := A,B)

The first two terms reduce to

(A<B → A,B := B−(B−A)/2, B−(B−A)/2)

For A=B, the third term becomes

(A=B → A,B := A,B) =
(A=B → A,B := B−(B−A)/2, B−(B−A)/2)

And the last term is

(A>B → A,B := A,B)= (A>B → ( ))

We have therefore shown that

$g1 \mid g2 \mid g5 \mid g6 =$
(A≤B → A,B := B−(B−A)/2, B−(B−A)/2) | (A>B → ( )) =
(A≤B → A,B := B−(B−A)/2, B−(B−A)/2) | ( ) =
f

orem, a program (for example, Append_con) will often compute a value in a domain larger than necessary (for example, domain(Append_abs)). Thus, we actually want to show

$r \circ \text{Append\_abs} \subseteq \text{Append\_con} \circ r$

A verification methodology

We have seemingly developed two mechanisms for designing programs: (1) a functional model for showing the equivalence of a design and its implementation and (2) a commuting diagram for showing correct data abstractions. However, both are complementary ideas of the same theory. For example, the concrete design comment for Append in the previous section is just a concurrent assignment translatable into a Pascal source program via the techniques described.

This leads to a strategy for developing correct programs:

(1) From the requirements of a program, develop the abstract data objects that are needed.

(2) For each object, develop abstract functions that may be necessary to operate on the abstract object.

(3) Using the abstract object and operations as a goal, design the concrete representation of the object and corresponding representation function.

(4) Design a concrete function for each corresponding abstract function.

(5) Show that the representation diagram commutes. That is, the concrete function does indeed implement the abstract function.

(6) Develop correct programs from each concrete function.

Note the order of steps 2 and 3. It is important to understand the abstract functions before designing the concrete representation, since the appropriate representation will depend greatly on the application. Consider the implementation of a *date* data object. Depending on the abstract functions required, the following are all feasible concrete representations:

(1) *Store as character string MM/DD/YY*. This is appropriate if the date is simply a unique tag associated with some data and has no other semantic meaning.

(2) *Store as <YY,DDD> where integer YY is the year and integer DDD is the day of year*. This is quite efficient if sequential dates are needed.

(3) *Store as number of days since some initial date*. This is most efficient to compute distances between two days, avoids certain problems such as accounting for leap years in all functions, but is cumbersome to print out in its usual format.

(4) *Store as <MM,DD,YY> for integers MM, DD, and YY*. Computation on dates is a bit more cumbersome, but conversion to its usual printed form is quite easy.

The importance of this technique is that it can be applied at any level of detail. This article obviously considered only short program segments. For larger programs, only concepts critical to the success of a program need to be formalized, although a long-range goal would be to develop this or other techniques that could be applied to very large systems in their entirety. Its

Design example

The second example involves developing a while loop for the following specification:

$f(x,y) = (x>100 \rightarrow x,y:=x,x+1) \mid (x,y:=x,y)$

To develop this program from its specifications, use the four-step process based on rules D.I through D.V (explained in the main text). First determine if f is realizable by a while loop.

D.I. Is $\text{range}(f) \subset \text{domain}(f)$?

Since f is defined for all input, $\text{domain}(f)$ includes all values of x and y. $\text{Range}(f)$ is some subset of (x,y), so condition D.I is true.

D.II. For $(x,y) \in \text{range}(f)$, do we have an identity function, that is, $f(x,y) = (x,y)$?

There are two cases for x: $x>100$ and $x \leq 100$. For the case of $x>100$, we have from the specification that $(x,x+1) \in \text{range}(f)$, and $f(x,x+1) = (x,x+1)$, which is an identity. For the case where $x \leq 100$, we know from the specification that $f(x,y) = (x,y)$.

D.III. Find [b] that evaluates to true in $\text{domain}(f) - \text{range}(f)$ and false in $\text{range}(f)$.

Find a predicate b that is false on its range and true elsewhere. Since we want y to take on the value $x+1$ or x to be less than or equal to 100 on the range of f, we know that $(y=x+1)$ OR $(x \leq 100)$ will be true on the range and hence false on $\text{domain}(f) - \text{range}(f)$. So the negative of this has our desired property: $\text{not}((x \leq 100) \text{ or } (y=x+1)) = (x>100) \text{ and } (y<>x+1)$. Since the loop will exit when this predicate is false, $b = (x>100)$ and $(y<>x+1)$, giving the partial solution

```
while (x>100) and (y<>(x+1)) do
  {d}
```

D.IV. Develop d so that all values are preserved for f.

To find a function [d] that preserves the values needed for f, y needs to become $x+1$. So let $d = (x,y := x,x+1)$. Our solution is now

```
while (x>100) and (y<>x+1) do
  {x,y:=x,x+1}
```

or just

```
while (x>100) and (y<>x+1) do
  y:=x+1
```

D.V. Show that the loop must terminate.

We know that b is false on the range of the while statement. Thus, if we can prove that the loop terminates, the current values of x and y when the loop terminates must be a feasible solution.

major difference from other verification techniques is that it forces the programmer or designer to consider the functionality of the program as a whole, and it requires the designer to design data structures with operations that operate on those structures. Since this is central to the data abstraction model of program design, this technique is quite applicable to current thinking about programming.

The technique presented here was quite manual, with the development of trace tables that grow in complexity as the number of conditionals increases. However, much of the process can be automated. For example, most of the details in a verification proof consist of keeping track of the various trace table executions. But this is a mechanical, syntactic property of programs, and a computer is ideal for carrying out such repetitive tasks. At the University of Maryland, we implemented an extension called FSQ to the Support integrated environment to facilitate such proofs.[8] The goal is to develop a semiautomatic system that guides the user into making the correct decisions. This should greatly ease the problems in developing such proofs.

Program verification — whether using this functional approach or some other approach, like axiomatic or algebraic correctness — is not an easy task. However, programming is not easy, and the need for correct programs is great. Using the functional correctness method described in this article will not guarantee simplicity in developing large correct programs, but it does provide a methodological basis for developing correct programs.

The method described in this article adds to the current set of techniques addressing the important, but extremely difficult, problem domain of program verification. The software engineering field still has a long way to go before program verification becomes an accepted activity in all programming developments. This article simply describes another tool that can be evaluated along with the others in determining the best approach towards good engineering of software. ■

Acknowledgments

I thank Victor Basili and John Gannon for their helpful comments on earlier drafts of this article, and also thank the referees who greatly improved its quality.

Partial support of this work was obtained from Air Force Office of Scientific Research grant 90-0031 to the University of Maryland.

References

1. C.A.R. Hoare, "An Axiomatic Basis for Computer Programming, *Comm. ACM*, Vol. 12, No. 10, Oct. 1969, pp. 576-580, 583.

2. J.V. Guttag and J.J. Horning, "The Algebraic Specifications of Abstract Data Types," *Acta Informatica* Vol. 10, No. 1, Jan. 1978, pp. 27-62.

3. H.D. Mills, "The New Math of Computer Programming," *Comm. ACM*, Vol. 18, No. 1, Jan. 1975, pp. 43-48.

4. J.D. Gannon, R.G. Hamlet, and H.D. Mills, "Theory of Modules," *IEEE Trans. Software Eng.*, Vol. 13, No. 7, July 1987, pp. 820-829.

5. H.D. Mills et al., *Principles of Computer Programming: A Mathematical Approach*, William C. Brown, Dubuque, Iowa, 1987.

6. H.D. Mills et al., "Mathematical Principles for a First Course in Software Engineering," *IEEE Trans. Software Eng.*, Vol. 15, No. 5, May 1989, pp. 550-559.

7. E. Dijkstra, "On the Cruelty of Really Teaching Computer Science," *Comm. ACM*, Vol. 32, No. 12, Dec. 1989, pp. 1,398-1,404.

8. M.V. Zelkowitz, "Evolution Towards a Specifications Environment: Experiences with Syntax Editors," *Information and Software Technology*, Vol. 32, No. 3, Apr. 1990, pp. 191-198.

Marvin V. Zelkowitz is a professor of computer science at the University of Maryland, College Park, with appointments in the Department of Computer Science and the Institute for Advanced Computer Studies. His research interests are in software engineering, programming environments, measurement, and compiler design.

A member of ACM and a senior member of IEEE, he is a past chair of ACM SIGSoft and of the Computer Society's Technical Committee on Software Engineering. He obtained a BS in mathematics from Rensselaer Polytechnic Institute and MS and PhD degrees in computer science from Cornell University.

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN 31261, SAUDI ARABIA

COMPUTER ENGINEERING DEPARTMENT

The Computer Engineering Department seeks applications for faculty positions at all levels. Preference will be given to experienced applicants at the associate and full professorial ranks. Applicants must hold a Ph.D. Degree in Computer Engineering or related areas. Individuals with demonstrated research records and teaching experience in one or more of the following areas will be considered: Fault Tolerant Computing, Data Communication and Computer Networks, VLSI and Design Automation, Robotics, Computer Architecture. Teaching and research at the Department are supported by a VAX 11/7800, a fully equipped, Computer Graphics Center, as well as a University Data Processing Center that has AMDAHL 5850 and IBM 3090 mainframes. In addition, research and teaching laboratories in the department includes: Design Automation Lab, Digital System Design Lab, Microprocessor Systems Lab, Printed Circuit Board Facility, Robotics Lab, and Computer Communication Networks Lab.

KFUPM offers attractive salaries commensurate with qualifications and experience, and benefits that include free furnished airconditioned accommodation on campus, yearly repatriation tickets, ten months duty each year with two months vacation salary. Minimum regular contract for two years, renewable.

Interested applicants are requested to send their Curriculum Vitae with supporting information not later than one month from the date of this publication, to:

DEAN OF FACULTY AND PERSONNEL AFFAIRS
KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DEPT NO. 9053
DHAHRAN 31261, SAUDI ARABIA

MITRE

More than 200 of the nation's most advanced systems engineering projects have one thing in common:

The expertise of MITRE.

Because of the importance of their mission, every agency we deal with demands excellence, plus uncompromising performance. That's why they all look to The MITRE Corporation for systems engineering solutions that work . . . and that consistently meet or exceed their high expectations. As a result, we are currently working on over 200 vital, state-of-the-art systems engineering projects for over 50 government clients, including many of this nation's defense and civil agencies.

MITRE is assisting the U.S. Air Force in programs dealing with OTH backscatter radars, and other radar-based surveillance systems such as North Warning and AWACS, major command and intelligence information systems like SPADOC and IDHS, key communications capabilities like MILSTAR, JTIDS and GWEN, and a wide range of tactical control and support systems like Joint STARS. We are also working with the U.S. Navy on significant artificial intelligence projects; with the Army to establish secure voice communications, automated message handling, and data processing support; and with the DCA to investigate advanced radio frequency concepts for C^3 missions. Finally, our civil systems engineering efforts include the development of electronic filing and information distribution systems for the SEC; support of the Next Generation Weather Radar (NEXRAD) for the NOAA and FAA; and assistance in many toxic substance-related areas for the EPA.

For over 30 years, MITRE's expertise in systems engineering analysis, research, design, and evaluation has been internationally recognized and highly respected—and with good reason. Almost two-thirds of our technical staff have earned advanced degrees. And, just as important, their exceptional performance consistently delivers the systems engineering solutions our clients need—with the quality they demand.

If you have demonstrated uncommon talent and technical ability, bring them to a clearly superior environment: MITRE.

U.S. Citizenship required.

Communications

System Design and Analyses • Digital Communications • Microprocessor Applications • Communications Processors (Hardware and Software) • Local Area Networks • Protocol Development/Evaluation • Modulation/Coding Techniques • Secure Communications • Systems Test and Evaluation • Switching and Control Systems • Satellite and Terrestrial Transmission • Survivability/Vulnerability and Electromagnetic Pulse • Line of Sight, Tropo, Millimeter Wave and Fiber-Optics

Command and Control

Air Defense Systems (Deployable and Fixed) • Tactical Air Traffic Control Systems (Deployable and Fixed) • Foreign Air Command and Control • Missile Warning Systems • Satellite Surveillance Systems • Systems Analyses and Specifications • Software Development • Display Technology • Bus-Oriented System Architectures

Computer Systems

Requirements Analyses • Systems Analyses • Systems Architectures and Engineering • Technology Assessment • System Acquisition • Performance Analyses • Simulation and Analytical Modeling • Artificial Intelligence • Image Processing • Fault Tolerant Systems • Local Network Design • Ada • Software Cost Estimation • Computer Security • Software Metrics • Distributed Data Base Systems • Program Verification

Systems Architecture

Advanced Systems Design • Advanced Planning • Intersystems Engineering • Functional/Operational Analyses • Systems Inter-Operability • Cost Analyses

Radar Systems

Digital Signal Processing • Radar System Surveillance Techniques • Radar System Analyses • Radar System Design • Anti-Jam Analyses • Intelligence Operations Analyses • Embedded Computer Hardware and Software

System Software

On VM/CMS • MVS/ESA • *UNIX*® • VAX/VMS • DOS • OS/2 • Macintosh OS• Design, Implement and Document System Software • Performance Monitoring • Software and Hardware Evaluations • Enhance User Facilities • Applications Software—Corporate, Financial & Administrative Applications; MVS/ADABAS Environment; PL/1 & NATURAL Programming Languages

Software Technology

Knowledge Based Expert Systems • Automatic Programming • Fault Tolerant Systems • Reusable Software

Software Engineering

Performance Simulation • Real-Time Computer Software Analysis and Sizing • Ada Compiler Evaluation • Software Cost Estimation • Project Management Tools • Artificial Intelligence

UNIX is a registered Trademark of AT&T Bell Laboratories.

An Equal Opportunity Employer M/F.

BOSTON

For C^{3}I Systems Engineering opportunities, please send your resume to:

David H. Gentes
The MITRE Corporation
4811 Middlesex Turnpike
Bedford, MA 01730

WASHINGTON

For C^{3}I and Civil Systems Engineering opportunities, please send your resume to:

M.C. Mason
The MITRE Corporation
7525 Colshire Drive
McLean, VA 22102

The Reactive Keyboard: A Predictive Typing Aid

John J. Darragh, Ian H. Witten, and Mark L. James
University of Calgary

The Reactive Keyboard is a device that accelerates typewritten communication with a computer system by predicting what the user is going to type next. Obviously its predictions are not always correct — otherwise users could go away and let the terminal do their work for them — but they are correct often enough to support a useful communication device. The predictions are based on what the user has already typed during the present session or previous ones. Thus the interface conforms to whatever kind of text is being entered: English, French, Pascal, operating system commands, and so on.

Previous implementations of the Reactive Keyboard proved most useful in enhancing the command interface to the Unix operating system by predicting commands, arguments, and file names (see the sidebar on pp. 44-45).[1] In contrast, the version described here is designed for entering free text and runs on any Apple Macintosh computer with at least 1 megabyte of memory (though 2 megabytes are preferred). It makes full use of the Macintosh mouse-and-window environment to give users convenient control over display and acceptance of predictions, and it requires no physical keyboard.

The Reactive Keyboard generates predictions from a text model that is created and maintained adaptively from previously entered text. The modeling technique was developed for text compression and in fact forms the core of one of the most effective known compression methods (first described by Cleary and Witten[2]; see Bell, Cleary, and Witten's book for a recent survey of the field[3]). The modeling mechanism builds large, tree-structured models of characters in context; to make predictions, the adaptive prediction mechanism consults these models for matches to the current context.

A new interface that partially automates typed input increases the speed and ease of communicating with a computer, especially for physically disabled people.

To make this modeling technique suitable for use in a communication aid, we had to address two issues: First, whereas in text compression the adaptive prediction mechanism feeds an encoder that generates a bit-stream representing the message, for interactive use we had to equip it with a human interface that allows predictions to be displayed and selected. Second, we had to solve a number of problems of resource consumption — adaptation and retrieval time and storage space — to make the mechanism practical for use in an interactive personal computer environment.

The Reactive Keyboard does not depend on the kind of text being entered and can be used for a variety of purposes. It is unlikely to help a skilled typist, except perhaps in entering documents, such as legal contracts, that contain a preponderance of standard boilerplate paragraphs. But moderately skilled typists will find that it assists with the highly structured text commonly found in interactive dialogues and formatted data entry, and novice and reluctant typists (including children and the aged) will appreciate the help it gives in free text entry. Its adaptive mechanism combines in a uniform way many features that exist separately in interactive interfaces, such as short versions of command names, command and file name completion, user-definable abbreviations for words and phrases, menus of common operations, and brief command files. Being menu-based, the Reactive Keyboard can replace a physical keyboard in click-and-drag interfaces that also require some text entry. Finally, and most importantly, the Reactive Keyboard has already found application as a user interface for physically disabled people who have difficulty using a regular keyboard.

0018-9162/90/1100-0041$01.00 © 1990 IEEE

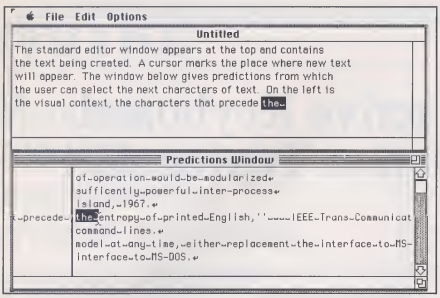

File Edit Options
Untitled
The standard editor window appears at the top and contains
the text being created. A cursor marks the place where new text
will appear. The window below gives predictions from which
the user can select the next characters of text. On the left is
the visual context, the characters that precede the␣
Predictions Window
of␣operation␣would␣be␣modularized␣
sufficently␣powerful␣inter-process␣
island,␣1967.␣
the␣entropy␣of␣printed␣English,''␣␣␣␣IEEE␣Trans␣Communicat
command␣lines.␣
model␣at␣any␣time,␣either␣replacement␣the␣interface␣to␣MS-
interface␣to␣MS-DOS.␣

Figure 1. The Reactive Keyboard, showing a word being selected in the predictions window (bottom) and copied into the editor window (top).

Using the Reactive Keyboard

The Reactive Keyboard is embedded in an editor for the Macintosh computer. Figure 1 shows a typical view of the screen. The standard editor window appears at the top and contains the text being created. The window below gives predictions from which the user can select the next characters of text. On the left is the visual context, the characters that precede the current position in the editor window. On the right is a menu of predictions offered as suggestions of how the context might continue. The user enters text by choosing one of these predictions and clicking at a particular point within it. Characters up to that point are inserted into the upper window, and the context and predictions in the lower one are updated accordingly — the context moves on and the predictions change completely.

Some users may find that the constant need to scan for correct predictions distracts them from the task at hand. However, the predictions can be ignored and text entered directly from the keyboard. In any case, after new text is entered, both context and predictions are updated as if they had been selected with the mouse. Thus, one can use the keyboard in the normal way until useful predictions appear.

Each prediction begins with a different letter, to maximize the chance that at least one letter can be entered without scrolling the prediction window. Successive characters in a prediction represent increasingly speculative guesses as to how the context will continue. The extent to which these are likely to be useful depends very much on the character of the text being entered. The completion of an English word, for example, will probably appear as the beginning of a prediction. Subsequent words in the prediction are less likely to be correct, except in stock phrases such as "one of the," "as well as," "the United States," and "in order to." Partway through a longer boilerplate phrase, many words will be enterable in one selection, and the next few predictions might be entirely correct too. Examples are "in the year of Our Lord, one thousand nine hundred and" in a legal document, "the Government of the United States of America" in a political document, and "latex paper.tex; lpr −d −Palw2 paper.dvi; lpq −Palw2" in an interactive dialogue.

The prediction window is scrollable and always contains 128 predictions, so that any ASCII character can be entered and users can dispense with a physical keyboard. Predictions are ordered by probability: the one beginning with the most likely next letter comes first. The user can control the number of predictions shown by making the window longer or shorter with the standard Macintosh mechanism for altering window size. Likewise, the length of each prediction can be controlled by making the window wider or narrower. Of course, the longer a prediction, the less likely it is to be grammatically correct. The vertical line separating context and predictions can be moved so that the length of the context displayed in the prediction window can be controlled too.

As the mouse is moved around over the prediction window, characters of the current prediction, up to the mouse position, are highlighted automatically, so that one can always see exactly what would be entered if the mouse button were clicked — for example, *the#* in Figure 1. (In the text of this article, # represents a space; in the figures a space is represented by the symbol ␣.) Thus, toying with the mouse causes the highlighting to change rapidly, providing a kind of two-dimensional selection capability. (We believe that in this interface there is ample justification for violating the normal Macintosh convention that changes occur only on mouse clicks.) The line of visual context in the prediction window moves up and down with the mouse so that it is always alongside the highlighted entry; the user sees the context leading into the current prediction. Also, that prediction is copied into the text in the editor window at the insertion point, providing additional selection feedback. (However, this copying is done only when characters are being inserted at the end of the text, to avoid rapid redrawing of the text that follows.)

Figure 2 illustrates the entry of several words of text in a sequence of eight screen images. (For presentation in this article, the windows are small and are placed side by side.) First, the words *Reactive#Keyboard#* are entered. The initial two letters are taken from *Research*, and the updated context and new predictions appear at the right of the second screen image. At this point *active#Keyboard#* is entered with a single mouse click, and fresh predictions appear. Again, two words — *primed#with#* — are entered together. The fourth image shows the effect of moving the cursor back into the context part of the prediction window: Now the last few characters of context (*with#*) are highlighted and, when the mouse is clicked, are deleted from the text (and, of course, from the context too). The remaining illustrations show more words, some of them incomplete, being entered. The final result is that six words have been entered in eight selections from a four-item menu, includ-

ing one selection that was needed to delete an erroneously chosen word.

Although it is perhaps easier to envision situations in which characters are being entered at the end of the text in the window, as in Figures 1 and 2, the system works equally well when the cursor is in the middle of the text, as in Figure 3. Predictions are conditioned on the context preceding the cursor (*...#has#been#*), and accepting a prediction inserts new characters at the cursor position.

The line termination character is represented on the screen by the arrow icon. In the examples, predictions end when this character is encountered; the system makes no guesses about how the next line will begin. For some kinds of text, it is appropriate to predict past the end of the line. Such capabilities are controlled by the Options item in the menu bar at the top of the screen, which the user can click to reveal the preferences sheet of Figure 3. The user can switch off "Stop predictions at Return" to allow predictions to be continued into new lines. Similarly, the user can select "Show Space, Tab, and CR" to have predictions displayed with icons representing these characters. The other options shown in Figure 3 control features of the predictive model and are described under "Implementation issues" later in this article.

The remaining menu bar items provide editing features familiar to Macintosh users. "File" allows files to be opened, closed, saved, printed, and so on. "Edit" permits cutting, pasting, and text searching.

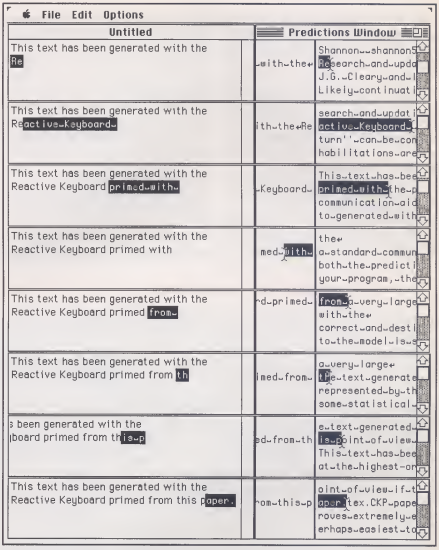


Figure 2. Eight views of the screen during text entry, showing text being selected in the predictions window (right) and copied into the editor window (left).

Adaptive predictive text generation

The ability to guess or predict future text relies on the statistical redundancy of language. C.E. Shannon estimated that English is about 75 percent redundant and noted that good prediction generally does not require knowledge of more than a fairly small number of preceding letters of text.[4] Although a native speaker's success in predicting English gradually improves with increasing knowledge of the preceding text, it does not improve substantially beyond knowledge of eight to 10 preceding letters.[4,6] The examples shown in the figures in this article were generated using at most seven letters to predict the next letter.

The adaptive model. A fundamental decision in the design of a predictive typ-

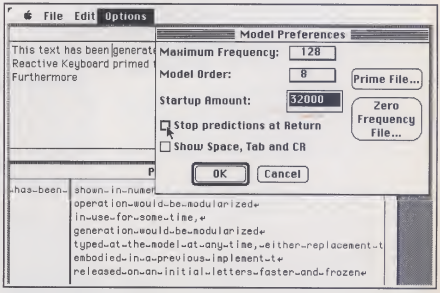


Figure 3. The Options menu.

ing aid is whether or not to make the text model adaptive. Many communication aids embody static models, created by the designer and frozen for the lifetime of the device. For example, these models might include standard abbreviations of common words and phrases thought to be representative of a typical user's input. Although such techniques are general-purpose and easy to implement, they lack flexibility and often constrain users with an unnecessarily inefficient language model. In contrast, dynamic models adapt to their user's vocabulary and phrasing.

Dynamic models can be constructed either explicitly by the user or implicitly by the system (Greenberg discusses these alternatives more fully[7]). User-defined abbreviation facilities, an example of explicit modeling, illustrate its chief drawback: Users must anticipate future language usage and divert themselves from the task at hand to create the model. Implicit modeling systems construct their models behind the scenes by monitoring input and updating the internal model accordingly, employing limited cues and rules to guess likely user behaviors. The models they form are generally probabilistic, and predictions may be inaccurate — particularly if user errors are incorporated as if they were part of the user's intended behavior.

The Reactive Keyboard adopts a dynamic, implicit, adaptive modeling strategy. It works by suggesting, on the basis of preceding inputs, what the user might want to select next, exploiting stored information about past selections to predict future ones. The device identifies likely continuations by locating a prediction context of recent selections in a large memory of previously encountered element sequences. In other words, predictions are made on the basis of the present situation, represented by short-term memory, and past experience, represented by long-term memory. The model is continually updated and adjusts itself automatically to the individual user's linguistic idiosyncrasies. The prompting display changes after each selection to present a new menu of predicted elements.

A keyboard-oriented, predictive terminal emulator

The modeling technique that forms the basis of the Reactive Keyboard has been used in a previous implementation that enhances the command interface to the Unix operating system. This program, called RK-Button, is used as a terminal emulator with a standard VDT and keyboard. It is a general-purpose command-line editor with the addition of predictive text generation. It communicates with a standard command interpreter, or "shell," and allows simple editing of input lines.

RK-Button opens the Unix shell and sends input to it through the "pipe" mechanism. Any output from the shell is simply directed to the screen. A consequence of this architecture is that programs need do nothing special to work with RK-Button. The predictive interface automatically switches off when a program — for example, a screen editor — is competing for control of the display.

An interaction with Unix by means of RK-Button is shown in part (a) of the figure. The predicted characters are written in reverse video on the screen and represented in the figure by shaded rectangles. Control characters are preceded by ^, and ^J is the end-of-line character. The column on the right shows the keys actually struck by the user.

Part (b) of the figure gives the meaning of a few of the control keys; in fact, many more line-editing features are provided. (The choice of control characters used originates from a PCD Maltron one-handed keyboard. This choice may seem unsystematic; however, these functions are invariably bound to function keys on the terminal for convenience.) Although not illustrated in the figure, the system is set up so that typing noncontrol characters simply overwrites the predictions; thus one can use the keyboard in the ordinary way without even looking at the screen.

Part (a) of the figure shows the entry of five command lines. Within each of the five groups preceded by the $, each line actually overwrote its predecessor on the screen. Consider the first group. After the first prediction, *mail*^J, the user struck ^N to show the next one. This prediction, *cd#news*^J, replaced the previous prediction on the screen. The user accepted the first word of it, *cd#*, by using the ^W command, moved to the next prediction, and accepted it in its entirety. The only thing remaining on the screen at that point was *cd#rk/papers/ieee.computer*. Then the next three commands were predicted in their entirety, while the last one required four keystrokes. The screen contents at the end of the dialogue are shown in part (c) of the figure.

In summary, five command lines comprising a total of 138 characters were entered by means of 11 strokes on just three function keys — an average of 2.2 keystrokes per command line or 12.5 characters per keystroke. This is fairly typical of command-line dialogues with Unix. The average time for generating a prediction is 12 milliseconds, and by default the system is initialized with 80,000 nodes of storage, which consume 1 megabyte.

RK-Button has recently been ported to the IBM PC to provide a similar predictive interface to MS-DOS. This version has been stripped down to reduce resource consumption, and program and model memory together occupy 80 kilobytes. The program responds to keystrokes instantly, without any perceptible delay.

Experience of Reactive Keyboard users. RK-Button has been in use for some time by two physically disabled students at the University of Calgary as their standard command interface to Unix. Their combined daily experience amounts to approximately four years. One student types with a single partially paralyzed hand and uses the system for entering commands and electronic mail. He estimates that over a two-year period it has provided assistance on more than 30,000 host system commands, averaging 10 predicted characters per command. He writes:

> "The Reactive Keyboard has dramatically changed the way I use computers. I now use much longer, more descriptive file names than I otherwise would have without its reliable recall and typing assistance. I also rely completely on RK-Button to remember such things as electronic mail addresses and long, complex command-line sequences. Life on line would just not be the same without it."

A four-phase cycle of *user selection*, *memory update*, *lookup*, and *display update* continues for the duration of a message composition session.

The adaptive model could be based on any level of redundancy present in natural language, ranging from the orthographic through the syntactic, semantic, and even pragmatic levels.[8] However, models are more difficult to generate and utilize at the higher levels, so the Reactive Keyboard, along with all other predictive text generation systems to date, uses a simple lexical model based on *n*-grams — consecutive sequences of *n* characters of text, where *n* is termed the "order" of the model. These, together with associated occurrence frequencies, are gleaned from representative text samples and from the user's input. The basic idea of prediction is to use the first *n*−1 letters — the prediction context — to predict the *n*th. The model's predictive power depends on how accurately its experience matches the user's current communication needs and on the number and length of stored sequences.

An important innovation in the Reactive Keyboard is that likely continuations for the current context are sought on a longest-match basis. If no elements in memory match the full prediction context, the context is truncated until a match is found. This technique proves extremely effective because it is capable of predicting from more complete, lower order models if less complete, higher order models lack instances of ti.e current context. All lower order models are implicit in the highest order model, so no extra storage is required. Such models have been found very effective for text compression.[2,3]

Model construction. The model is stored in a special tree structure that allows partial matches between context and model to be found economically. An example is shown in Figure 4a. Here, the incomplete phrase *to be or not t* is represented as a tree with maximum depth four. The children of the root — space (#), B, E, N, O, R, and T — are all the characters that have been seen

The other user has a progressive neuromuscular disorder. He uses RK-Button as a general command interface and writing aid and is otherwise unable to write. His comments:

"I find the Reactive Keyboard to be an extremely beneficial tool for typing. Since I have severe neurological damage in my hands, it seems to cut the time I spend coding manyfold. To illustrate this, I need only inform you how I mailed John this letter. All that was required was my typing *ma*, after which time it predicted *il darragh^J*. So, I was able to type and enter a command normally requiring 13 keystrokes in but three! This saves much time, since it is rather difficult for me to access some keys on the board. It must be remembered that both my hands and fingers move slowly and inaccurately."

The system has recently been released on the Usenet network, and the following unsolicited comment is typical of those received:

"I have cerebral palsy, so my typing is a little bit impaired. I am very impressed with your program. It seems to be ideal for significantly speeding up my typing in the shell while I am programming. I like it a lot, and it saves me a lot of time. I am very glad to have it available to me. The other people that I have showed it to are also very impressed."

(a)

```
$ mail^J                                                    ^N
  cd  news^J                                                ^W
  cd  news^J                                                ^N
  cd  rk/papers/ieee.computer^J                             ^L
  cd  rk/papers/ieee.computer

$ emacs paper.tex^J                                         ^L
  emacs paper.tex

$ rm  paper.tex.CKP paper.tex.BAK^J                         ^L
  rm  paper.tex.CKP paper.tex.BAK

$ wc −w paper.tex^J                                         ^L
  wc −w paper.tex

$ readnews −n comp.sources.unix^J                           ^N
  mail ^J                                                   ^W
  mail ^J                                                   ^N
  mail  bdarragh%uncamult.bitnet@ucnet.ucalgary.ca^J        ^L
  mail  bdarragh%uncamult.bitnet@ucnet.ucalgary.ca
```

(b)

Key	Description
^C	Accept the next predicted character
^W	Accept the next predicted word
^L	Accept the whole predicted line
^N	Show the next alternative prediction
^P	Show the previous alternative prediction

(c)

```
$ cd rk/papers/ieee.computer
$ emacs paper.tex
$ rm paper.tex.CKP paper.tex.BAK
$ wc −w paper.tex
$ mail bdarragh%uncamult.bitnet@ucnet.ucalgary.ca
```

Using RK-Button: (a) dialogue with Unix; (b) some commands; (c) screen contents at the end of the dialogue.

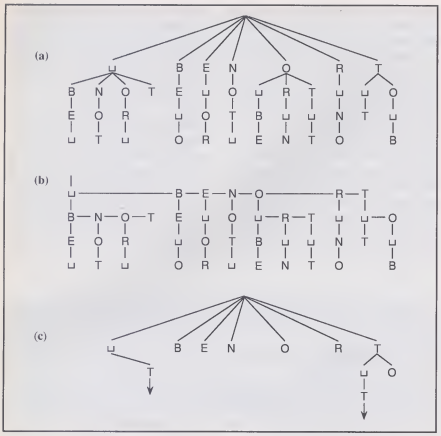


Figure 4. Tree memory structure: (a) in concept; (b) linked-list implementation; (c) generating predictions.

in the null context, that is, all characters that have occurred. The children of the # that roots the leftmost subtree — B, N, O, and T — are characters that have occurred in the context #, that is, at the beginning of a word. The child of the leftmost B, namely E, is the only character that has occurred in the context #B, and its child # is the only one that has occurred in the context #BE. The tree is rather stringy because the sample of text is very small.

The aim of the storage structure is to allow predictions associated with substrings of the context to be found quickly. A tree-structured scheme is essential to avoid repeated searches for each different context length. Figure 4b shows one way of storing the tree: alternatives at each level are linked together as a list. Each node has four parts: a character element label, a frequency counter (not shown), a pointer to the next alternative at its level, and a pointer to the first element for this context at the level below. Alternatives at each level are listed alphabetically for presentation in this figure only; in practice, frequency order is used to speed updating and menu generation.

A number of important technical issues arise in implementation. Models become quite large and must be carefully organized because of the real-time search and update requirements imposed by an interactive user interface. Moreover, the system must work continuously, with no upper bound to the amount of text that can be accommodated. Three operations are required to update and maintain the model: incrementing frequency counts of recurring n-grams, adding n-grams to the model as new input behavior is experienced, and removing n-grams to make way for new ones when the model is full.

The update procedure processes each input character as follows: First, if memory is full, up to n nodes are deleted to ensure that sufficient space is available for possible additions. (At most n free nodes will be needed to store an input character, one at each of n levels.) Then, at each level of the tree, the part of it associated with the current context is searched for the character. If the character is found, the corresponding node's frequency count is incremented. Finally, frequency order is maintained by resorting the linked list of alternatives to speed future access. If the input character is not found, a new node is created and added to the end of the list with a count of one.

Model consultation. The purpose of the model is to support prediction. Figure 4c shows the part of the tree relevant to the process of predicting the character that follows the phrase *to be or not t*. The prediction context is T#T. There are no predictions at this level (right arrow); nor are there any for the context T# (left arrow). When the context is further truncated to T alone, two predictions — # and O — appear in the rightmost subtree; truncation to the null string generates seven predictions — #, B, E, N, O, R, and T.

The Reactive Keyboard generates the initial characters of its predictions with a simple ordering strategy that favors longer contexts. First, any matches found for the highest order context are added to the menu, starting with the most popular. Then, progressively shorter matches are made and any new character predictions that have not yet been encountered are added to the list, again in frequency order. This ensures that each initial menu character is unique and limits the total number of items to the size of the ASCII alphabet. When the context length becomes zero, all remaining characters that have not yet been found but have occurred in the input stream are added in frequency order. Finally, all remaining characters are added to the list.

Figure 5 gives an example (adapted from a paper by Foulds, Baletsa, and Crochetiere[9]) of how the initial letters of predictions would be generated if the user were spelling the word *happy* and had already entered HAPP. It is based on a 26-letter alphabet plus space, with $n = 4$. The alternative predictions for each context length are shown in frequency order from left to right. Notice that the relative popularity of candidate letters varies with the length of the context used for prediction. For example, with three letters of context the most popular predictions are E, R, L, ..., in that order (first line in Figure 5); with no context they are #, E, T, A, ... (fourth line). Once a letter appears at a given level, it reappears at all lower levels; these occurrences are shaded in the figure because they do not represent new contributions to the set of predictions. Letters not previously encountered at higher levels are added to the menu list in frequency order. The final set of predictions, combining all contributions from the various

levels, is shown in order at the bottom of the figure.

Once the list of initial letters has been generated, each in turn is followed "into the future" to complete the menu item. The system makes progressively longer concatenations by assuming that the current prediction is correct and destined to be accepted. In the example in Figure 5, the prediction context APP would be shifted forward to make concatenated predictions. The process repeats with the projected new context until either the end-of-line character is predicted or the window width is exceeded. For example, if the current context APP predicted the initial letter Y, then Y would be assumed correct and the context shifted to PPY. This would be used to predict another character, say #, and the process would be repeated with the projected context PY#. If no further elements were predicted, the concatenation Y# would be displayed for selection.

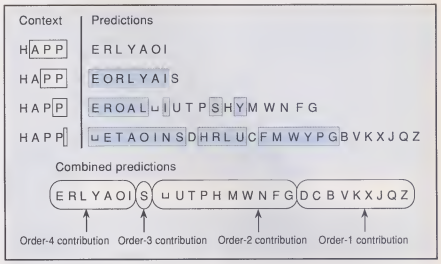


Figure 5. Generating initial letters for the menu.

Implementation issues

We have described the new communication medium that the Reactive Keyboard provides: adaptively predicted text, offered to the user for selection. Embedding it in a system raises some important practical considerations that greatly affect its usability. The success of the modeling technique hinges on the parameters of the model — the maximum order *n* and the amount of space allocated to the model — and on the text used to prime it.

Model size. The order of the model determines the depth of the tree represented in memory. Text compression experiments show that the model's predictive power is relatively insensitive to the order, providing it exceeds a certain minimum.[2] The Reactive Keyboard uses a default value of $n = 8$, although the "Options" menu (Figure 3) allows users to experiment with different values.

The size of the model is a second important parameter, since it dictates the total number of *n*-grams that can be accommodated. It is difficult to estimate accurately the minimum size required to store a complete model of an open-ended stream of text. In practice, limits are imposed by implementation considerations: the amount of storage available, the size of each tree node, and the type of data structure employed. The Macintosh version of the Reactive Keyboard uses the linked-list structure of Figure 4b (although we have experimented with alternative storage structures, such as a binary tree with three pointers per node) and has a default memory allocation of 1.5 megabytes, which accommodates 128,000 nodes. This seems to be adequate for normal purposes, but the user can allocate more or less if desired.

Priming the model. By far the most important factor affecting the quality of predictions is the text used to prime the model. The statistical characteristics of seemingly similar samples of English can vary greatly.[10,11] Predictions work best if the text used for priming is carefully selected to be as representative as possible of what the user wants to generate. Consequently, the Reactive Keyboard provides facilities for the user to choose the priming text and to change it when necessary.

On start-up the Reactive Keyboard primes itself automatically from a default file. The default can be changed with the "Prime File..." button in Figure 3. To avoid too great a start-up delay, only the last part of the priming file is used. The default amount is set to 32,000 characters but can be altered by the user ("Start-up Amount" in Figure 3). Priming the model with this amount takes a few seconds, but a thermometer-like display reduces the perceived delay significantly (despite the fact that the actual delay is increased about 20 percent by the overhead of updating the thermometer!). The user can reprime the model at any time, either replacing or augmenting the current model (accessed from the "File" menu). Of course, modeling also proceeds automatically from user inputs. The overall result is that the model is kept up to date, yet users have the option of explicitly specifying (and even altering) the text used to prime it. Since the model can be erased and reprimed, the user can either keep multiple contexts concurrently in the model, or — by clearing it first — switch between separate, possibly very different, global contexts, depending on what seems to work best.

The entire ASCII character set can be used as *n*-gram elements, but many characters never occur in normal text. Analysis of several fairly large ASCII files — both English and program source code — revealed that about 30 to 40 characters were not represented at all. Consequently, the priming file may fail to give initial frequencies to some characters. To overcome this, the system reads a standard (or user-defined) frequency table before starting a session ("Zero Frequency File..." button in Figure 3). This table contains a frequency-ordered list of the symbol set derived from a very large sample of representative text. It is used to build a backup zero-length context (in other words, context-free) level into the model. Symbols with equal frequency are listed in the table in collating order, so that any that do not occur in the source sample are presented last, in ASCII sequence.

Integrating with Macintosh software. As implemented, the Reactive Keyboard is embedded in a particular, rather rudimentary, text editor. Ideally, rather than being built into one application, it should be

modularized in a way that makes it usable with any interactive program, intercepting all input and passing the appropriate characters on to the application as though they had been typed at the keyboard. Communication is needed in the reverse direction, too, to communicate the context back to the predictor. If the Reactive Keyboard tried to maintain its context separately from the application, a degenerate form of operation would ensue, for although there would be no problem with continuous input, the context could not be updated properly when the user selected different points within the application's window. In this situation the context should be cleared when mouse clicks occur within the application's window, so that incorrect contexts do not generate misleading predictions as the user moves around within the text.

Even accepting the inevitability of this degenerate mode of operation, achieving the desired modularization is infeasible because the current Macintosh operating system does not provide sufficiently powerful interprocess communication facilities. One possibility would be to implement the interface as a "desk accessory," a kind of program that is always available to the user. This can be done with the current version of the Macintosh operating system (Version 6), provided that the desk accessory is loaded in the appropriate way to permit communication with application programs (that is, in the applications heap rather than the system heap). However, this creates problems if several applications are loaded at once (for example, using the popular Multifinder desktop control program) — a mode of operation that is becoming increasingly popular and may be the only one supported by new versions of the operating system.

The upshot of all these considerations is that although it would have been technically possible to implement the Reactive Keyboard so that it could be used with any application program, the nonstandard communication involved would probably not work with future versions of the Macintosh operating system. Consequently we were forced to adopt the much less satisfactory solution of a stand-alone editor to demonstrate the power of predictive interfaces.

For good typists the Reactive Keyboard is generally a hindrance (except for occasional text generation in a keyboardless graphics interface). As suggested earlier, fair to poor typists may find it worthwhile for highly structured text. People with no keyboard experience and people who are prevented by physical disabilities from using standard keyboards efficiently may view it as an indispensable writing aid. Of all potential users, those with severe physical limitations and communication disabilities stand to gain the most. Although the device described here is completely new and we do not yet have much field experience to go on, a predecessor that employs the same prediction technique with a keyboard-based interface has been operational for some time, and many people have found it a valuable time- and energy-saving device (see sidebar, pp. 44-45).

Earlier systems for predictive text generation fall into three main categories: letter anticipators, word or phrase completers, and combined systems.[1] Letter anticipators accelerate user input by making likely next letters faster and easier to select. Completers amplify inputs by offering words or phrases based on an initial prefix of one or more letters spelled by the user. Combined systems, as the name implies, do both: most use letter anticipation but switch to word/phrase completion if a known prefix is entered.

The Reactive Keyboard improves on these devices in three important ways. First, it uses adaptive modeling, basing its predictions on previously entered text instead of a standard dictionary, and it provides convenient facilities for priming with samples of text appropriate to the task at hand. Second, it concatenates predictions, each line in the menu being constructed character by predicted character, and takes advantage of two-dimensional selection to give users the opportunity to maximize the productivity of each selection. Third, its variable-length modeling technique offers higher quality predictions, as numerous text compression experiments have demonstrated.[3]

By combining these three improvements in a smooth user interface, the Reactive Keyboard represents a worthwhile advance over earlier systems. The beneficial effects of adaptive modeling are very striking, and although it is not possible to convey this in the brief examples presented here, user comments (see sidebar) indicate how helpful it can be. The attention given both to the adaptive modeling technique and to user interface issues is unprecedented in communication aid design. The Reactive Keyboard has great potential to enhance the ease and rate of communication, especially for physically limited people. ■

Availability of the Reactive Keyboard

The Reactive Keyboard is available in three versions: RK-Unix, RK-PC, and RK-Mac. The first two are implementations of RK-Button (see sidebar) and provide accelerated entry of command lines to the Unix and IBM DOS operating systems. The third is the implementation described in this article. All three versions are available in source form at no charge over Internet via anonymous file transfer from

```
cs-sun-fsa.cpsc.ucalgary.ca
[136.159.2.1]
```

in

```
pub/the.reactive.keyboard
```

Acknowledgments

This research has been supported by the Natural Sciences and Engineering Research Council of Canada and by the Alberta Heritage Foundation for Medical Research. We would like to acknowledge the enthusiastic support of David Hill throughout the work.

References

1. J.J. Darragh, *Adaptive Predictive Text Generation and the Reactive Keyboard*, MSc thesis, Computer Science Dept., Univ. of Calgary, 1988.

2. J.G. Cleary and I.H. Witten, "Data Compression Using Adaptive Coding and Partial String Matching," *IEEE Trans. Communications*, Apr. 1984, Vol. COM-32, No. 4, pp. 396-402.

3. T.C. Bell, J.G. Cleary, and I.H. Witten, *Text Compression*, Prentice-Hall, Englewood Cliffs, N.J., 1990.

4. C.E. Shannon, "Prediction and Entropy of Printed English," *Bell System Technical J.*, Jan. 1951, pp. 50-64.

5. T.M. Cover and R.C. King, "A Convergent Gambling Estimate of the Entropy of English," *IEEE Trans. Information Theory*, Vol. IT-24, No. 4, July 1978, pp. 413-421.

6. C.Y. Suen, "*n*-gram Statistics for Natural Language Understanding and Text Process-

ing," *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. PAMI-1, No. 2, Apr. 1979, pp. 164-172.

7. S. Greenberg, *User Modeling in Interactive Computer Systems*, MSc thesis, Computer Science Dept., Univ. of Calgary, 1984.

8. J.A. Pickering and G.C. Stevens, "The Physically Handicapped and Work-Related Computing: Towards Interface Intelligence," *Proc. Second Int'l Conf. Rehabilitation Engineering*, The Rehabilitation Eng. Soc. of North America, Bethesda, Md., June 1984, pp. 126-127.

9. R.A. Foulds, B.S. Baletsa, and W.J. Crochetiere, "The Effectiveness of Language Redundancy in Non-verbal Communication," *Proc. Conf. Devices and Systems for the Disabled*, 1975, pp. 82-86.

10. H. Kucera and W.N. Francis, *Computational Analysis of Present-Day American English*, Brown Univ. Press, Providence, R.I., 1967.

11. C.D. Gibler, *Linguistic and Human Performance Considerations in the Design of an Anticipatory Communication Aid*, PhD thesis, Northwestern Univ., Evanston, Ill., 1981.

John J. Darragh is a research associate in computer science at the University of Calgary. He received a BSc degree in biochemistry and computer science and an MSc in computer science, both from the University of Calgary. His MSc work specialized in human-computer interaction, and the Reactive Keyboard was his thesis project. He has presented his work at a number of conferences in North America.

Ian H. Witten is a professor of computer science at the University of Calgary, where he served as head of computer science from 1982 to 1985. Previously, he was a senior lecturer at Essex University, England. His current interests include prediction and modeling, machine learning, programming by example, and autonomous systems. The author of four books and numerous papers, he received a degree in mathematics from Cambridge University, a degree in computer science from the University of Calgary, and a degree in electrical engineering from Essex University. He is a member of the IEEE Computer Society.

Mark L. James is a research associate at the University of Calgary, where he was responsible for the development of the Reactive Keyboard on the Macintosh. He has also been involved in the implementation and maintenance of the Unix version of the Reactive Keyboard. A Macintosh consultant and programmer, he also has extensive experience with Unix-based machines.

Readers may write the authors at the University of Calgary, Dept. of Computer Science, Calgary, Canada T2N 1N4.

Become a vital part of U S WEST Advanced Technologies, one of the country's technology leaders. We're an integrated team of research, development and operations professionals, making technology work for U S WEST's business units and their customers.

We work mainly in the information processing and telecommunications fields, developing and engineering new products, services and systems for our parent company. Our applied research group focuses on advanced software technology, artificial intelligence and advanced user interfaces, while also investigating emerging technologies to bring into the corporation.

If you're an entrepreneurial professional with an eye on the big picture, we'd like to hear from you. We continue to offer opportunities for individuals with B.S., M.S. or Ph.D. degrees in Computer Science or Electrical Engineering.

Send your resume to us at: **U S WEST Advanced Technologies, 6200 S. Quebec Street, Suite 340, Englewood, Colorado 80111.**
We are an equal opportunity employer.

Offers of employment are conditional on applicants undergoing and passing a pre-employment drug test, subject to applicable federal, state and local law.

U S WEST®
ADVANCED TECHNOLOGIES

©1989 U S WEST, Inc.

Preliminary CALL for PAPERS

OOPSLA '91

Sixth Annual ACM Conference on

Object-Oriented Programming Systems, Languages, and Applications

6-11 October 1991, Phoenix, Arizona

Program Committee

Alan Snyder, HP Labs (chair)
Grady Booch, Rational
Luca Cardelli, DEC SRC
Pierre Cointe, Rank Xerox
William Cook, HP Labs
Peter Deutsch, ParcPlace Systems
Gail Kaiser, Columbia U.
Henry Levy, U. Washington
Henry Lieberman, MIT
Mark Linton, Silicon Graphics
Ole Lehrmann Madsen, Aarhus U.
Ron Morrison, U. St. Andrews
Eliot Moss, U. Massachusetts
Mayer Schwartz, Tektronix
Guy Steele, Thinking Machines
Jacob Stein, Servio Corporation
Dave Thomas, Object Technology
Mario Tokoro, Keio U./Sony CSL
Chris Tomlinson, MCC
Dionysios Tsichritzis, U. Geneva
Jeannette Wing, CMU
Rebecca Wirfs-Brock, Tektronix
Aki Yonezawa, U. Tokyo

Program Chair

Alan Snyder
OOPSLA 91 Program Chair
Hewlett-Packard Laboratories
(150) Page Mill Road)
P.O. Box 10490
Palo Alto CA 94303-0969
+1-415-857-8764
oopsla91@hplabs.hp.com

Conference Chair

John Richards
OOPSLA 91 Conference Chair
IBM Research Center
P.O. Box 704
Yorktown Heights NY 10598
+1-914-784-7731
oopsla91@ibm.com

Object technology is influencing many areas of computer technology, including programming languages, software engineering, user interfaces, databases, distributed systems, and OSI network management. The availability of practical support for applying object technology is encouraging its use in real-world software development.

The annual OOPSLA conference is the premier forum bringing together researchers, developers, and practitioners to share ideas and experiences related to object technology. Areas of interest include: language design and implementation, tools and environments, components and frameworks, principles and theory, concurrent and distributed systems, methods and processes, databases and persistence.

Papers

The conference will include both invited and contributed papers. Authors are encouraged to submit high quality papers describing relevant research or experience. **Research papers** should describe work whose purpose is to advance the state of the art of object technology. **Experience papers** should describe work in which object technology has been applied for some purpose. The program committee will evaluate each paper on its relevance, clarity, correctness, originality, and significance. Special consideration will be given to promising experience papers. Papers must not be under consideration elsewhere (or published) in the same or similar form. Do not submit multiple papers describing essentially the same work.

Authors should send **six** copies of the full paper, in English, to the program chair to be received no later than **1 March 1991.** Papers must be limited to 18 pages, typed double spaced or typeset 10-point on 16-point spacing. A separate cover sheet must contain contact information (contact name, postal address, and phone number), a 100-word abstract, and indicate the paper category (research or experience). *Submissions failing to meet these conditions will be rejected.*

Authors will be notified of acceptance or rejection by **20 May 1991.** Accepted papers prepared on special forms are due **20 July 1991**. Authors of accepted papers are expected to sign an ACM copyright release form and present the paper at the conference. Proceedings will be distributed at the conference and via *SIGPLAN Notices* and will be available from ACM Press. Outstanding papers may be considered for a special issue of a journal. Awards will be given to the best paper in each category.

Guidelines for authors will be published in the Fall 1990 issue of *OOP Messenger*, or can be obtained from the program chair.

Experience Reports

Space will be made available in a separate track for short presentations of unrefereed reports describing practical experience applying object technology to real-world, product-quality software development. Prospective speakers should submit a 1-2 page description of the project scope and status and the specific points to be covered in the presentation to the program chair by **1 April 1991**. Selection will be based on relevance and potential interest. Summaries will be collected after the conference for publication in *OOP Messenger*.

Other Contributions

For information on tutorials, panels, workshops, demos, and vendor exhibits, see the Call for Participation, which will be available in October at the 1990 conference or from the conference chair.

Important Dates

1 March 1991
Papers due
1 April 1991
Experience reports due
20 May 1991
Acceptance notification
20 July 1991
Camera-ready papers due

Papers due 1 March 1991

Knowledge Bases and Databases

Converging Technologies, Diverging Interests

Yehudah Freundlich
Data General

Computers are tackling new and exciting tasks, from advising insurance underwriters to monitoring processes. They diagnose, plan, and schedule. They provide unprecedented flexibility in office automation, design, text retrieval, and many other domains. These applications transformed the way we think about computing and brought a new perspective to programming — a knowledge perspective. They engendered a new breed of programmer — the knowledge engineer — who develops knowledge bases. They introduced a new type of database, an *expert* database that stores not only values but "chunks" of knowledge.

In supporting these new applications, practitioners of knowledge-based and database technologies are finding their traditional areas of interest overlapping. The technologies are working together to implement expert databases and large, persistent, distributed knowledge bases. In this sense, the technologies are converging. Their fundamental interests, perspectives, and research directions, however, remain divergent.

Knowledge-based technology brought the perspective of knowledge to applications programming and database modeling. But KB specialists focus on the knowledge level, whereas database specialists focus on the data level.

Knowledge-based programming

Consider first the perspective of knowledge-based-systems programmers. Knowledge engineers are fundamentally applications programmers. They construct programs that use stored information to monitor processes, diagnose malfunctions, adjudicate insurance claims, and so forth — or they are concerned with what it takes to build such applications. What differentiates a programmer of knowledge-based systems from a conventional programmer is not that the knowledge-based-systems programmer writes programs that embody knowledge and reasoning. All programs embody knowledge and reasoning of some sort; otherwise they could not perform as they do. Even a simple program to update a bank account and keep track of accumulated interest has knowledge: It knows how to calculate interest and update the account.

Knowledge-based-systems programming differs from conventional programming primarily in its methodology, the core of which is a concern for making explicit the knowledge embedded in the program. Explicit means open to direct manipulation. Within the programming context, this means removing the knowledge from the procedural setting in which it is usually embedded in conventional programming and representing it in declarative form.[1]

Making knowledge explicit is not merely an organizing principle for coding — not just a matter of centralizing variables or grouping them in structures. It rests on a distinction between two types of knowledge. Knowledge can be embodied in a program as a procedure or as a data structure. This distinction corresponds to the philosophical difference between knowing *how to* and knowing *that*.

Consider an accounting program that contains procedures for updating the account, say, semiannually. Now, the interest period can be "compiled" directly in the procedure with an instruction for updating the account semiannually, or it can be stored as a variable to be accessed when needed. The program works correctly either way.

0018-9162/90/1100-0051$01.00 © 1990 IEEE

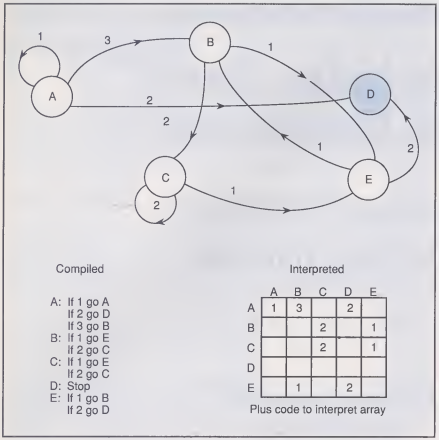


Figure 1. A state-transition graph exemplifying compiled versus interpreted knowledge.

In the first case, however, knowledge about the interest period is embodied as a procedure; the system knows how to update the account correctly, doing so every half year. In the second case, this knowledge is represented declaratively; the system, in addition to knowing how to update the account, knows that the interest period is semiannual. In this case, knowledge about the interest period is more explicit.

Figure 1 depicts two versions of a state-transition system. The compiled version, written in pseudo-Basic, knows how to simulate the state transitions. The interpreted version knows, in addition, what the state transitions are. Figure 2 depicts two versions of a Prolog program for printing colors.[1] The compiled version knows how to print the colors; the interpreted version also knows what the colors of snow, grass, and sky are.

```
Compiled version
   printcolor(snow)      :– !, write("It's white.")
   printcolor(grass)     :– !, write("It's green.")
   printcolor(sky)       :– !, write("It's blue.")
   printcolor(X)         :– write("Beats me.")


Interpreted version
   printcolor(X) :–      color(X,Y), !, write("It's "), write(Y),
                         write(".")
   printcolor(X) :–      write("Beats me.")

   color(snow, white)
   color(grass, green)
   color(sky, blue)
```

Figure 2. Printcolor as an example of compiled versus interpreted knowledge.

A knowledge-based program contains a knowledge base — a set of data structures viewed as a set of statements about the world.[1] All programs probably have some declarative knowledge. Still, conventional programmers think procedurally. Systems analysts map a problem domain to process specifications, dataflow diagrams, or entity relationship models. Conventional programmers embody these as flow charts in the program. There is no direct map back from the program to the problem domain.

A knowledge-based program is more explicit about the objects in the world and how information flows between them, and about the states of knowledge and how they change. A knowledge engineer constructing an automobile diagnostic system considers such statements as "Fuel clogging is a possible malfunction" along with statements about the possible and necessary symptoms of fuel clogging. For an expert system dispensing financial advice, the knowledge engineer may deal with assumptions regarding governmental attitudes toward regulatory agencies. As an analyst, the knowledge engineer wants to clarify these matters. As a programmer, the knowledge engineer wants to make them explicit in the program — to set up data structures that map directly back to these statements.

Conventional system design involves a functional decomposition ensuring that the program will do what it is supposed to. Knowledge engineers must, in addition, select statements in the problem domain to be made explicit and must construct (find, use) a language to express those statements.[1] (Rule-based and object-oriented programming are two general programming paradigms knowledge engineers use to model the problem domain. Knowledge engineers mix and match these as well as other programming paradigms.)

Knowledge engineers, however, are not mere encyclopedists recording knowledge. The knowledge so meticulously recorded must be used by a program. The primary use of knowledge in knowledge-based systems is reasoning, broadly defined as the accessing of information implicit in the knowledge base.[1,2] In a diagnostic system, for example, information about the system to be diagnosed can be seen as containing

implicitly the information about what is causing the system to malfunction, or about what further information is needed to determine the cause of the malfunction. Similarly, in the accounting program, the data concerning the account's current state, the interest period, the interest rate, the current date, and so on, implicitly states (if we know how to derive it) what the current account should be updated to. The code that does the reasoning is the inference engine. This is the code that uses information in the knowledge base to derive other information implicit in the knowledge base.

Knowledge-based systems thus contain a knowledge base and an inference engine. Though this dichotomy has been stressed within the context of rule-based programming, it has nothing to do with rules per se. It is the knowledge engineer's way of distinguishing between knowledge of what is (statements about the world) and use of that knowledge for a particular purpose. In Figure 1 the knowledge base is the array (and perhaps variables expressing dynamic information, such as the current state and the current input). The array is a way of (a language for) expressing state transitions as a function of input. Code feeding off the array to simulate transitions is an inference engine using the knowledge base for simulation. The knowledge base for the system in Figure 2 is the set of Prolog relational statements that give the colors of snow, grass, and sky. The inference engine uses this knowledge to print their colors.

This methodology has many benefits — for example, understandability. We understand knowledge more readily in declarative form than when embodied as a procedure in a particular context. This is because the meaning of a statement, as opposed to the meaning of an instruction, is relatively free of its context. To understand what an instruction does, we must understand the procedure in which it is embedded; a statement can be understood without reference to how it is used. (This, of course, is only relatively true.)

Another benefit is a greater degree of modularity; it is easier to delete or modify part of the system without affecting the rest. A statement's validity is relatively independent of its context; modifying other statements or procedures won't change its correctness. On the other hand, any programmer knows how easy it is to cause procedural bugs in working parts of the system by modifying other parts. Modularity and understandability facilitate maintainability. Indeed, knowledge-based-systems methodology often makes it easier to create tools that let nonprogrammers maintain the system and modify the knowledge themselves.

One major advantage of this methodology is extensibility, or reusability. Knowledge removed from a particular procedural context is free to be used in many contexts, within the existing application and in others, often in ways not foreseen when developing the original application. A model of a refinery constructed for diagnostic purposes may be extended for monitoring process control, perhaps to supply advice for a plant shutdown. In the accounting program, the interest period for updating bank accounts, if stored declaratively, also can be used for financial projections. The state-transition system of Figure 1 can be used with additional code and information to make probabilistic predictions about halting times. With supplementary code, the color knowledge base of Figure 2 could be used to color scenes rather than merely to print colors.

Table 1. Four attempts to distinguish between a database and a knowledge base.

Differentiator	Database	Knowledge Base
Collector	Clerk	Expert
Use	Retrieval	Multiple
Type of information	Facts	Higher level
Theoretical requirements	Computational theory	Semantic interpretation

Knowledge bases versus databases

All this may seem very familiar to database specialists. Consider the points made above:

(1) A separation should be made between data and applications using the data.

(2) Data needs to be understandable, modular, and maintainable.

(3) Separating data from applications allows the data to be used by many applications.

Surely, this is what data modeling and database management are all about: storing information for many purposes in a form that many users can access, understand, and modify.

Knowledge bases may not seem to differ from databases. Both are data structures, or sets of data structures, in which information is stored and from which information is retrieved. Yet intuitively, knowledge bases and databases are not quite the same. Consequently, many researchers have sought to distinguish between data — that which resides in databases — and knowledge — that which resides in knowledge bases. Table 1 shows four attempts to distinguish between knowledge bases and databases.[3-5]

In the first attempt, when an expert (someone with specialized knowledge) is needed to collect the information, that information is knowledge and is stored in a knowledge base; however, when a clerk can do it (no specialized knowledge is needed), the information is data and is stored in a database. The second differentiator says that databases are used only to retrieve information, whereas knowledge bases have many uses: to inform, to diagnose, to monitor, and so on. The third differentiator is based on the type of information stored: Facts, such as a description of employees working in a company, are data; higher level abstractions — for example, a statement that an elephant is a type of mammal — are knowledge.

According to the fourth attempt, the bases have different theoretical requirements. A knowledge base has a semantic interpretation, that is, a specification of its relationship to the real world. For example, in a rule of the form "If A then B," we must stipulate whether "then" means causal consequence, typical conjunct, or logical necessity. A database, on the other hand,

must be provided with a computational theory for realizing the base on physical machines.

In a sense, these attempts, taken collectively, are correct; this will be discussed later. Taken at face value, however, they cannot be rigorously maintained. Considerations regarding the collector or the use of the information are extrinsic to the data structures themselves. The collector's identity has no bearing on whether the information can be stored in a database, and information once retrieved can be put to any use whatsoever, whether it's from a knowledge base or a database; it can be reasoned with and used to inform, to explain, to advise.

Second, the distinction between facts and higher level abstractions, or between that which requires no interpretation and that which does, cannot be rigorously maintained. Any abstraction or higher level generalization is a fact — for example, that an elephant is a kind of mammal is a fact. A relational database can easily store information about species, genus, and so forth. Finally, it would hardly do to say that databases come without interpretation. A company would be very disconcerted indeed to know that the answer to its query about the number of employees earning more than $50,000 has no relation to reality.

The question of whether a set of data structures is a knowledge base or a database is thus misdirected — like asking if it is hotter in the city or in the summer. There is no intrinsic difference between a knowledge base and a database. The difference is one of perspective.

To view a set of data structures as a database is to be concerned with data-level issues: What data structures are used to store the information? What mechanisms are used to manipulate and access these structures? Are they efficient? Do they meet performance standards? How is the data's integrity maintained? What mechanisms allow for concurrency?

To view the structures as a knowledge base is to be concerned with knowledge-level issues[3,6]: What do the structures tell us about the world? What kinds of knowledge are represented? (For example, can we express incomplete knowledge, say, that some employee works with John but we don't know who that employee is?) What is the scope? (What counts as a company product?) What is the granularity of the knowledge (how fine a distinction is made between different products)? What can the system know at any time? What follows from what it knows? (If John is listed as Bill's manager, will the system conclude that John is an employee?)

A set of data structures used by a program is both a knowledge base and a database.

This is not to say that a database offers no answers to knowledge-level questions. A database must have scope and granularity. A database has some semantic interpretation: it says something about the world. A database management system must allow for some elementary inferencing such as numerical comparisons and counting. (Thus, implicit in the company employee database is information about how many employees have salaries greater than $50,000.) In this sense, all databases are knowledge bases.[3]

Traditional databases, however, are primitive when viewed from the knowledge level. Their expressive capabilities are weak; they are limited to positive instances of predicates and relations of individuals, and they exclude disjunction, negation, and quantification.[1,3] Their inferencing capabilities are restricted to retrieval and the elements of reasoning already mentioned — numerical comparisons and counting. The semantic interpretation of traditional databases is obvious to potential users. Questions of how the data is to be obtained and what may be done with the information bear little relevance to any database programming effort. Responsibility for obtaining, supplying, and using information stored in a database lies with users — expert or otherwise — and not with the database. Thus, traditional database specialists have shown little concern for knowledge-level questions. Databases *are* knowledge bases, but database specialists have not been overly concerned with this fact.

Conversely, knowledge bases are databases, though knowledge engineers have not paid much attention to this fact. We can ask data-level questions of a functioning knowledge-based system. The knowledge is stored in data structures, mechanisms retrieve the information and meet performance requirements, and the knowledge base is maintained and managed. In this sense, any knowledge base is a database.

From the database perspective, however, knowledge bases are primitive. They are not particularly large, and they tend to be bound to applications; therefore, they are not as public and as persistent as databases. Extending the use of a knowledge base has been mostly a matter of extending the expert system capabilities or generating a new expert system with its own knowledge base — thus the lack of concern with data-level questions. When plagued with inefficient system performance, knowledge engineers usually react with ad hoc "fine-tuning." Maintainability has been treated only at the knowledge level — ensuring that the knowledge in the system is kept up to date. Knowledge engineers only marginally considered traditional database concerns of security, storage, and integrity.

In sum, a set of data structures used by a program is both a knowledge base and a database; it says something about the world and is implemented on a physical machine. To delve more deeply into its expressiveness — what it says, implies, allows to be said — is to adopt a knowledge base perspective. To ask about implementation mechanisms is to view it as a database. The domain of database technology has not required expressive capabilities as sophisticated as those of knowledge-based technology. Hence, database specialists have delved less into knowledge-level questions. On the other hand, knowledge base specialists have required less sophisticated implementation mechanisms, and so have been less concerned with data-level issues.

Converging technologies

Database and knowledge-based technologies can no longer ignore each other. As their domains expand, their interests overlap. If knowledge-based-systems technology is to reach its potential, large, shared, distributed knowledge bases are needed. As computers tackle new tasks in new domains — CAD/CAM, VLSI design, imaging, text retrieval, office automation, distributed network computing — support is needed for large-scale, complex, data-intensive applications in user-oriented environments. Handling these tasks requires a new way of thinking about databases, a shift from the view of data as values — sets of uniformly formatted data types — to a view of data as chunks of knowledge.

Knowledge bases as databases. Consider the needs of knowledge-based-system technology. First, many domains require very large knowledge bases. Coopers and Lybrand estimate that for an aluminum rolling mill, 4,000 objects and 3,000 rules are needed to embody the 500 pages of reference manuals for one alarm and sensor system; 180,000 objects are needed for the entire 15,000 pages of reference manuals. For a chemical process reactor, 4,000 objects and 1,200 rules were required to embody 200 pages of procedure manuals covering just two reactors and two products. Second, to exploit the potential advantages of extensibility and reusability — the fact that the knowledge base, or parts of it, can be put to many uses — knowledge bases will have to be shared among several applications, just as databases are.

Finally, an important advantage of knowledge-based systems is that they can integrate various types of expertise distributed throughout an organization, expertise that must be brought to bear simultaneously to make a decision. To do this, a single inference engine would have to use diverse knowledge bases stored, perhaps, in several locations and maintained by different people.

Large, shared, distributed knowledge bases lead to traditional database problems of security, integrity, concurrency, storage, retrieval, and efficiency. Knowledge-based-system specialists look to database technology for help in solving these problems. However, this requires a paradigm shift by database researchers, a change in the way they view data — one they have been edging toward since the 1970s, for reasons of their own.

User-oriented environments. The driving force for this paradigm shift is the trend from machine-oriented to user-oriented computing, from batch processing for specialized users mediated by specialized programmers to highly interactive, flexible, on-line systems used by diverse communities. Users, often unsophisticated as programmers, are accessing and integrating an increasingly broader range of processing functions. They need an easy-to-use interface: a consistent, uniform, and simple pattern of interactions flexible enough to accommodate users as individuals and as members of diverse communities.[7,8]

This trend has profound implications for database technology. First, distinctions between database management, applications programming, and end use must be eliminated, or at least blurred. Second, *metadata* — information at the schema level (constraints, views, relations, and so forth) — must be treated as regular data. Third, databases must capture inferential relationships. All this adds up to a shift in the conception of data — of what is stored in databases.

Consider some user-oriented environments that databases support. An image database management system handles thousands of intermediate-level descriptions, called tokens, generated by researchers programming out of necessity. Researchers generate tokens on the fly by grouping, splitting, or modifying existing descriptions. These descriptions must include their relations to other tokens. Individual researchers create, delete, and restructure a database, write complex experimental applications in the database language, and examine the results on a graphics screen.

In a hypertext environment, a document is no longer a fixed sequence of text, but a set of text (or media) objects that a reader can structure and restructure as desired, within limits set by the author. A text document is no longer a flat file but a database of objects with link and web relations. Authoring a hypertext document defines the database structure and stores the objects in the database. Reading a document activates webs and links, and it defines, on the fly, a database view. From the database perspective, both reader and author are end users.

In a distributed network computing environment, such as Data General's distributed applications architecture, users choose the most appropriate components — workstations, applications, networks, operating systems, databases, I/O devices, and so on — for their computing needs and integrate them accordingly. Network managers distribute resources such as applications, databases, and printers and modify the distribution as new resources and resource types are added or deleted, or as the user base changes. They arrange access levels for the various users and user types in accordance with current security needs. End users then integrate these resources. A user might apply statistical package *A* to analyze database *B*, file the result as document *C* in directory *D*, print it on printer *E* using the presentation package *F*, and mail it to person *G* — all without regard to the physical node location of these resources or to format conversion and vendor compatibility. Information about the resources and users (resource types, accessibility, and so forth) is maintained in a database. Network managers and end users are not database specialists.

Expert databases. In user-oriented environments — and they are proliferating — users need modeling capabilities traditionally mediated by systems analysts and database managers. They need more than this — more than just tools for doing what was traditionally database management, and more than the ability to specify a data dictionary. They need to treat the objects in the data dictionary as regular data, as objects of discourse.

Users access, manipulate, and modify data dictionary objects in an integrated manner with data in the database. In systems design and analysis, for example, users reason explicitly about constraints.[9] In the imaging environment discussed above, tokens generated by researchers need to be persistent, shared, machine-readable data; otherwise, researchers will constantly re-implement the same data structures and manipulation procedures. A network manager, an end user from the database perspective, creates and deals with different kinds of resources, sundry security considerations, and diverse types of users with various relationships between the types. In these environments, users treat information about structure as information, or metadata as data. Metadata gives users information they need to accomplish their tasks. It must be preserved and shared just like data values in traditional databases.

It is easy enough to store information about structure in traditional databases. We can store the fact that an elephant is a kind of mammal as a relation in a relational database; fields can be tacked on to records that state constraints on other fields; links between document objects in a hypertext document can be specified as link relations. But this is mere syntax, like my planner-organizer that has a place for listing things to do but never does them for me; it merely provides a slot for storing values. The purpose of structuring is to facilitate inferencing, or the drawing of implications. If users are to have access to structure, they must also have access to the implications of structuring; that is, they must be able to capture inferential relationships. In other words, the database system itself must provide some semantics.[10] The database system supplies semantics when

(1) stipulating that an elephant is a kind of mammal ensures that all records of type elephant inherit appropriate properties from the type mammal;

(2) storing a range of users in a field called "accessibility constraints" in a record of type resource constrains the accessors of the resource to the users stipulated; and
(3) stipulating links for a hypertext object in a hypertext database links the given object to the other objects specified.

Furthermore, the user must have some control over the semantics, that is, the ability to modify types of inferencing and to specify new types of inferential relationships. A system modeler may want not only to specify that an elephant is a type of mammal, but also to define different types of "is-a" inferences: a definitional is-a, an assertional is-a, and a typical is-a. (A definitional is-a might be "a square is a quadrilateral"; an assertional is-a might be "a bank is a self-interested institution"; a typical is-a might be "a mammal is a land animal.") The modeler may want not only to specify constraints on resource accessibility by explicitly listing users, but also to specify constraints in terms of user types, or even place conditions on the constraints, perhaps by rules. For example, if such and such conditions obtain for someone, this resource (or this type of resource) is not accessible to them.

The four suggested characteristics of knowledge bases cited in Table 1 are symptomatic of this new kind of database. First, traditional databases do have a semantic interpretation, but it is the user or analyst who does the interpreting. For example, we interpret an employee record in the database as referring to an employee. The new databases, on the other hand, have built-in semantics: The system itself draws the implications of an elephant's being a kind of mammal. Second, traditional databases store values, and the system does little more with these values than allow us to retrieve them. Built-in semantics, however, might relate to reasoning, explaining, informing, or whatever. The meaning of a constraint statement, for example, is that it constrains. (Although constraint statements constrain in traditional databases also, in those cases they are metastatements; they are part of the data dictionary and are not themselves data.)

Third, traditional databases do not store higher level facts, meaning that the system itself does not draw the necessary implications. Storing the fact that an elephant is a kind of mammal in the new type of database, however, affects an entire class of facts in the database, making this a higher level fact. Finally, users in a user-oriented environment are knowledgeable in the domain of use; they understand the concepts and know how to reason with them.

All this spells out a new concept of what a database stores — not just values, but organized sets of values, where (1) the structure is itself accessible and can be manipulated by a user, (2) the structure has implications inferred by the database system itself, and (3) the user or modeler can define what sorts of inferences the database is entitled to make. In other words, what is being stored is a chunk of knowledge. Let's call this type of database system an *expert* database system.

(An object-oriented database is a particular kind of expert database. In object orientation, the chunks are viewed not as passive objects merely to be stored, retrieved, and manipulated, but as active objects. The stored object has procedural and declarative knowledge. The database user makes requests of an object, and it responds on the basis of its knowledge. Object-oriented databases have begun to use a standard set of system-defined inferential relationships — aggregation, generalization, and so forth — with varying degrees of built-in semantics. The database designers have also settled on particular mechanisms for access and for user-defined semantics — namely, methods and handlers.)

With the advent of expert database systems, knowledge-level questions can no longer be ignored. Scope, granularity, semantics, the organization of knowledge, expressivity, and types of knowledge and their representation are now of genuine concern to the database specialist. In this sense the technologies are converging. There is a need, on the one hand, for large, persistent, distributed knowledge bases, and on the other for databases that store chunks of knowledge.

Diverging interests

Nevertheless, the interests of knowledge base specialists and database specialists differ, and their research directions diverge. They even preach contrary strategies: Knowledge base technology urges separation of the knowledge base from the inference engine; expert database technology promotes the capture of inferential relationships in the database. The two technologies differ fundamentally in their underlying assumptions: A database system is intended for a diverse set of users; a knowledge-based system supports only one community of users, however broad-based that community may be.[9]

The knowledge engineer assembles a complex body of knowledge for use by a relatively specialized community. This requires a high degree of expressivity within the user domain, linked with correct and timely inferencing. Faced with a fundamental trade-off between expressiveness and computational tractability,[1] the knowledge engineer structures and assumes. Classifying, constraining, and linking facilitate reasoning by directing inferencing strategies. Making assumptions, as in "any case is a nonworst case," limits the search space. These measures limit expressivity, since whatever does not fit into the scheme is inexpressible.

Knowledge engineering consists of designing a suitable trade-off — of finding structures, assumptions, and corresponding inferencing strategies that provide a good balance between expressivity and inferencing capability. The task of knowledge representation research is to explore the trade-off spectrum: What are the various types of structures, assumptions, and inferencing strategies? How do they limit expressivity? Which are appropriate for specific domains and tasks?

More specifically, some of the central issues in knowledge representation research have to do with kinds of knowledge, with logics (systems of inferencing principles), and with reasoning strategies. We can have knowledge about objects, processes, goals, motivation, causality, time, actions, performance, and so on. We can have meta-knowledge (knowledge about knowledge and knowledge about how to reason to obtain implicit knowledge). Knowledge can be essentially incomplete (for example, existential knowledge). How do we represent these different types of knowledge? Are there other kinds? Which concepts are primitive? Which derivative? What are their relationships?

There are also many kinds of logics: first-order logic, nonmonotonic logic, fuzzy logic, modal logics (dealing with beliefs, necessity, and so forth), and nonformal logics (for example, analogic). Which are appropriate? Are they (or how do we make them) computationally efficacious? How do we formalize them within a computer program?

Much is known — and still more unknown — about reasoning strategies. Researchers have cited

- taxonomic reasoning (via inheritance in a classification system),
- default reasoning ("in the absence of information to the contrary, assume..."),
- goal-oriented (backward) reasoning,
- data-directed (forward) reasoning,
- the establish/refine strategy (establish a set of candidates and refine the set), and
- constraint propagation (following the implications of a constraint along a network).

There is a host of other strategies as well. How do we represent the knowledge so that the strategies are computationally tractable? How do we make the strategies explicit in the program? What implicit assumptions do these strategies rely on (for example, the closed-world assumption that everything relevant is explicitly stated)? These questions are central to knowledge representation research.[1]

The database specialist stores knowledge for diverse users. Implementation is the primary focus. Concurrency, query optimization, data structures and algorithms, performance evaluation, database systems reliability, database machines, database management system architectures, security, and distributed databases[3] are primary issues. Database research, even in semantic data modeling and expert databases, continues to focus on implementation details, albeit for the new type of databases that store chunks of knowledge rather than mere values. For example, database specialists seeking to store spatial and temporal information in a knowledge-oriented database discussed the problem in terms of indexing and hashing, clustering for sequential access, and so forth.[3] However, a knowledge base researcher dealing with temporal information considered whether time should be represented as discrete points or as intervals, and then sketched an interval-based temporal logic.[1]

In short, knowledge base specialists continue to focus primarily on knowledge-level issues and database specialists on data-level issues. The distinction is fuzzy at times because the issues cannot always be cleanly separated. Nevertheless, the knowledge engineer's paramount interest is not to store knowledge but to represent it so that it can be used by a relatively specialized community. The database specialist's paramount interest — even the expert database specialist working on semantic modeling — is not semantics but how to store semantics for diverse users.

The way we think about computing has been transformed. Knowledge-based technology has brought the perspective of knowledge to applications programming. First, programs do not merely do something, they embody knowledge about the world. Therefore, this knowledge should be made explicit, encoded in data structures as statements about the world. Second, we can view these data structures at the knowledge level by considering what they say about the world and what they allow to be said.

Database technology has also adopted the knowledge perspective. A datum stored in a database is not merely a value but a chunk of knowledge, a structured set of values with inferential relationships to other chunks of knowledge.

Knowledge-based and database technologies will have to work together. For knowledge-based technology to achieve its potential, knowledge base specialists must take traditional database concerns seriously. And database specialists, to implement expert databases on machines, need to understand what a chunk of knowledge is. In this sense, the technologies converge. Nevertheless, their interests and research directions diverge. The knowledge base specialist focuses on knowledge-level questions: What sorts of knowledge chunks are there? Which are appropriate for specific domains? The database specialist focuses on data-level issues: how to define, store, retrieve, and manipulate different kinds of knowledge chunks under the constraints of a working database system. ■

Acknowledgments

I would like to thank those who helped me clarify my ideas through discussions and with comments on previous versions of this article: Steve Petrofsky and Bill Wright of the AI Marketing Group at Data General, Howard Glassman, Avron Barr, Mark Shirley, and, of course, the referees.

References

1. R.J. Brachman and H.J. Levesque, eds., *Readings in Knowledge Representation*, Morgan Kaufmann, Palo Alto, Calif., 1985.

2. A. Barr and E.A. Feigenbaum, eds., *Handbook of Artificial Intelligence*, Vol. 1, Addison-Wesley, Reading, Mass., 1981.

3. M.L. Brodie and J. Mylopolous, eds., *On Knowledge Base Management Systems*, Springer-Verlag, New York, 1986.

4. M.L. Brodie, J. Mylopolous, and J.W. Schmidt, eds., *On Conceptual Modeling*, Springer-Verlag, New York, 1984.

5. G. Wiederhold, "Knowledge and Database Management," *IEEE Software*, Vol. 1, No. 1, Jan. 1984, pp. 63-73.

6. A. Newell, "The Knowledge Level," *AI Magazine*, Vol. 2, No. 2, Summer 1981, pp. 1-20.

7. I. Grief, ed., *Computer-Supported Cooperative Work*, Morgan Kaufmann, Palo Alto, Calif., 1988, pp. 609-622.

8. L. Bic and J.P. Gilbert, "Learning from AI: New Trends in Database Technology," *Computer*, Vol. 19, No. 3, Mar. 1986, pp. 44-54.

9. L. Kerschberg, ed., *Proc. First Int'l Workshop Expert Database Systems*, Benjamin/Cummings, Menlo Park, Calif., 1987.

10. W.D. Potter, "Traditional, Semantic, and Hypersemantic Approaches to Data Modeling," *Computer*, Vol. 21, No. 6, June 1988, pp. 53-63.

Yehudah Freundlich was AI applications manager in the AI Marketing Group at Data General, providing knowledge engineering and consulting services to support the design, development, and implementation of knowledge-based and object-oriented systems. He co-chaired the Second Annual Conference of the International Association of Knowledge Engineers in October 1990. Earlier in his career he held faculty positions in physics at Tel-Aviv University and in philosophy at Ben-Gurion University.

In addition to articles on knowledge-based systems, Freundlich has published papers on physics and on such philosophical topics as the structure of scientific theories, the criteria of evidence, the nature of explanation, theory evaluation, and methods of scientific inquiry. He earned his BS in mathematics and physics from Brooklyn College in 1960, his AM in physics from Columbia University in 1962, and his doctorate in theoretical elementary particle physics in 1969 from the Technion, Israel Institute of Technology. He is a member of the IEEE Computer Society.

Readers can write to the author at Data General Corp., 50 Maple St., Milford, MA 01757.

Sixth Annual Conference on COMPUTER ASSURANCE

SYSTEMS INTEGRITY, SOFTWARE SAFETY, AND PROCESS SECURITY

CALL FOR PAPERS

IEEE

COMPASS is dedicated to finding ways to assure that computer systems perform correctly under all circumstances within the context of the systems they control. COMPASS is a contraction for COMPuter ASSurance. COMPASS addresses the requirements, methodologies/techniques, and framework for assured computing. Computer systems must satisfy requirements that include safety, real-time constraints, security, integrity, fault tolerance, and correctness. Methodologies/ techniques include formal verification and validation, reliability, quality assurance, and systems engineering. The framework addresses issues of philosophy, policy, human factors, risk containment/mitigation, cost-benefit tradeoffs, and certification.

Papers and panels are solicited. Topics include:

- *Formal methods*
- *System worthiness/ certification*
- *Education in computer assurance*
- *Measurements (metrics)*
- *Assurance in medical and nuclear power systems*
- *Software certification for EEC '92*

Papers and panel proposals should be double-spaced, single-sided only. Five copies are to be sent.

- *Submissions Due:* *11 January 1991*
- *Acceptance Notices:* *28 February 1991*
- *Conference Dates:* *25-27 June 1991*

LOCATION: **National Institute of Standards and Technology (NIST) Gaithersburg, Maryland (suburban Washington, D.C.)**

COMPASS Sponsors: **IEEE Aerospace & Electronics Society**
IEEE National Capital Area Council

Information:
Dolores R. Wallace
NIST
E-Mail: wallace@swe.ncsl.nist.gov
Phone: (301) 975-3340

Send Papers to:
Program Chair: Roger U. Fujii
C/O Logicon, Inc.
222 W. 6th Street
San Pedro, California 90731
E-Mail: R.Fujii@ieee.org
Phone: (213) 831-0611, ext. 2420

Strategies for Supporting Application Portability

James D. Mooney, West Virginia University

A major portion of the software development effort for many current application programs is devoted to making those programs work satisfactorily in multiple environments. Despite the dominance of a few machine architectures and operating systems for some classes of applications, our industry continues to design and use many other systems.

Most developers cannot afford to limit applications to a single environment or develop them afresh for each new one. Methods are needed to adapt new and existing programs to a variety of environments and operating systems, both within a narrow range (such as different versions of Unix) and over much wider ranges (such as between micros and mainframes).

The need for portability can impose requirements on the design of both the programs to be ported and the system software components with which they interact. This article discusses a range of system design strategies that can be used to support portability and examines the ways in which these strategies have been employed by past and present systems. The discussion is meant for system designers and application designers and surveys a wide range of strategies. However, since space precludes an exhaustive treatment of all issues, I will place the principal emphasis on operating system issues.

Despite the growing demand for ways to transport programs to new environments, the literature devoted to portability is still limited. The lecture notes edited by Brown[1] provide a good foundation for study. A few more recent texts treat portability as a major topic. For the most part, however, discussions of portability in the literature are limited to case studies.

Many strategies are available to improve application portability. System designers and application designers must work together to produce the best solutions.

Application portability

Application portability is increasingly cited as a desirable goal in software development. But, there is disagreement about the exact meaning of this phrase. In this article, I will use the following as a working definition:

> An application is portable across a class of environments to the degree that the effort required to transport and adapt it to a new environment in the class is less than the effort of redevelopment.

The word *application* rather than *program* is used because the goal of portability is most useful and achievable for programs whose purpose is largely independent of the type of system on which they run. Most programs in this class are directed to end-user objectives rather than system management; they are application, not system, programs. This class can include general-purpose programs such as compilers and editors, as well as programs focused on a narrower application domain.

The word *environment* refers here to all elements of a total system with which an application explicitly interacts, either during development or during execution. These elements typically include an operating system (OS), language processors, procedure libraries, hardware units, and users.

The primary goal of portability is to facilitate the activity of *porting* an application from an environment in which it currently operates to a new or target environment.

This activity has two major aspects:

(1) *transportation* — physical movement of the program's instructions and data to the new environment, and

(2) *adaptation* — modification of the information as necessary to work satisfactorily in the new environment.

0018-9162/90/1100-0059$01.00 © 1990 IEEE

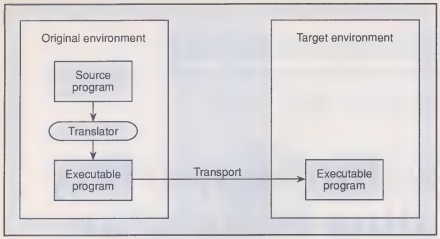


Figure 1. Porting a program with binary portability.

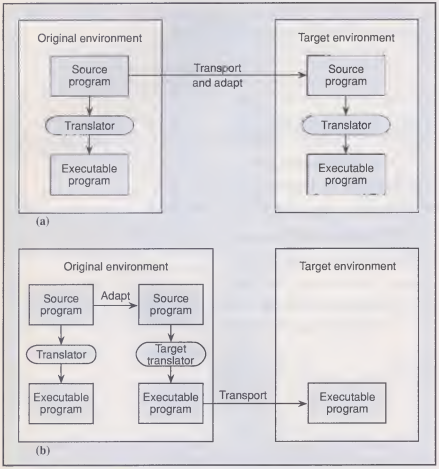


Figure 2. Porting a program with source portability: (a) procedure for standard target environments, and (b) procedure for limited target environments.

Problems of physical transportation include use of compatible media or communication channels, and interpreting and translating file formats, character codes, data representations, etc. Adaptation involves higher level modifications that might be necessary to adjust the program to work with aspects of the new environment that are intentionally or unavoidably different from the old one. Some adaptation can be done automatically by translation programs, including reprocessing the program's source code with a different compiler. Other types of adaptation must be carried out manually.

My definition also emphasizes that portability is a matter of degree. Desirable as it might be, porting an application to a significantly different environment with no changes whatsoever is seldom possible. Instead, the goal is to make it possible to adapt the program to the new environment with much less effort than rewriting the program from scratch. There might be a trade-off between the degree of portability and the class of target environments; if this class is restricted to a small set of very similar environments, a much higher level of portability might be attainable.

The portability that can be achieved in a particular case is determined both by the structure of the application and by the properties of each component of the target environment, including hardware, languages, and operating system. Several types of portability can be identified as potential goals for the designers of an application.

Binary portability. A program that can be transported in its executable form to a new environment and executed in that environment without change exhibits binary portability. No compiler or linker is needed in the target environment, and application source code is not required for porting. Binary portability must be perfect, since it is not feasible to modify the executable form of a program.

This type of portability is an attractive goal, since the effort of transporting the program is minimized. Users can install programs with binary portability in new environments without access to their source code. Binary portability can also enable direct sharing of executable program files over networks.

However, binary portability requires strong similarities among the target environments. The following elements at least must be identical:

(1) The CPU architecture (as viewed by the machine-language application program).

(2) The operating system calls used by the program.

(3) The memory organization as viewed by the program, including special locations such as interrupt vectors.

(4) The program's view of any I/O devices used.

Figure 1 illustrates the process of porting a program with binary portability. The figure indicates that no translators are

needed in the target environment; only the executable form of the program must be transported, and no adaptation is required.

Source portability. Programs that cannot achieve binary portability across a given set of environments can still achieve a high degree of source portability. This type of portability exists if the source files containing the program representation in its original form (usually a high-level programming language) can be transported to the target environment and made to work in that environment with only limited effort. This effort will always include recompilation and linking, using tools available for the new environment. Ideally, no modification of the source files would be required; in practice, some modifications are almost always necessary. This adaptation, however, should require much less effort than rewriting the program for the new environment from scratch.

Whereas binary portability is possible only within a narrow range of highly similar environments, source portability can be achieved for a broader range of targets.

Figure 2a illustrates the usual process of porting a program with source portability. The figure shows that the source program must be both transported and adapted, and a translator for the source language must exist in the target environment. Figure 2b illustrates an alternate method. In this approach, the translator for the target environment runs in the original environment. The source files are adapted prior to translation, and the executable program is transported afterward. This method is used when the target environment is not a suitable environment for translation.

Experience portability. The two types of portability described so far are designed to support the movement of programs into new environments. Reducing the effort required by users and programmers to adapt successfully to new environments is also important. If the experience gained by these users and programmers in one environment can be reused in a new environment, then *experience portability* (also called *user portability* or *personal portability*) has been achieved between these environments. This type of portability can be maximized through consistency in user interfaces, file structures, software tools, and other elements of the environment visible to those who interact with it. Experience portability may be supported by using portable tools and portable system software to implement user interfaces and file systems.

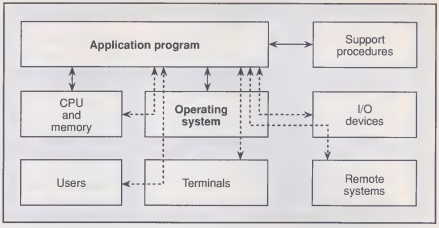


Figure 3. Application program interfaces. Solid arrows represent direct interfaces, and dashed arrows represent indirect interfaces.

Related concepts

A number of concepts broadly related to, but distinct from, application portability, have been identified as important goals for some types of software development. A full discussion of these concepts is beyond the scope of this article. However, to contrast them with portability, I discuss them briefly in this section.

Reusability. The goal of reusability is to reduce software development costs by designing general-purpose components, such as procedures or packages, which can be used in many applications. This leads to a "building-block" approach to software development, in which new applications can be constructed largely from high-level components already available. In contrast, the goal of portability is to allow reuse of complete existing applications in new environments.

Interoperability. A set of programs can cooperate in an important sense if they can conveniently share information, that is, if data or files produced by one program can be directly used and understood by another when this is logically meaningful. Important examples of data that programs might share in this manner include word processing documents, pictures, and spreadsheets. Interoperability is the term applied when programs are able to "operate together" in this way. The increasing need for interoperability has led to the development of common data formats for various special kinds of data and to the popularity of integrated environments in which a number of cooperating applications are combined in a single consistent system.

In general, interoperability refers to the sharing of data among different types of applications. This sharing can be achieved only if the applications agree on the same high-level data formats, or if specialized translators are provided to convert one form of data to another. A special case occurs, however, when data is to be shared among different instances of the same application that might be operating in different environments. These programs face the problem discussed in the next paragraph.

Data portability. When data is to be shared by the same program across environments, we have a need for data portability. High-level data formats will remain consistent, but a portability problem exists similar to that faced by the applications: the data must be transported and adapted to its new environment. In this case, the adaptation does not change the logical meaning of the data, but must account for the difference in system conventions for data formatting. This type of adaptation can usually be automated. Its requirements depend only on the conventions of the source and target environments, not on the specific applications that will use the data.

Strategies for portability

An application interacts with its environment through a collection of interfaces. Figure 3 illustrates the interfaces between a typical application and its environment at the time of its execution. It shows that the program has direct (visible) interfaces with three key components:

(1) The CPU and memory (through machine instructions).

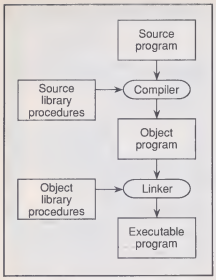


Figure 4. Program development and representations.

(2) Runtime support procedures (through procedure calls).
(3) The operating system (primarily through system calls).

These direct interfaces are shown by solid arrows in the figure.

In addition, a number of indirect interfaces (shown by dashed : rrows) link the application to a system component through a series of direct interfaces involving other elements. For example, the interface between an application and most I/O devices is mediated by the operating system. The operating system also provides an interface to memory management functions, and to certain CPU resources such as timing mechanisms. At a still higher level, the interactive user interface provided by many applications is mediated by both the operating system and the user's terminal device; and access to remote systems and resources on a network is routed through the operating system and communication devices.

Any of these interfaces might also be mediated by the support procedures; this variation is omitted from Figure 3 to avoid clutter. Note that all of the application's indirect interfaces pass through the operating system.

As stated, these interfaces represent a view of a program during execution. Additional elements of the environment have a role to play prior to execution. Figure 4 illustrates the role of the principal representations of a program that occur during its development and their associated translators. These representations are alternate forms of the direct interfaces between the program and its environment. There are additional opportunities to overcome interface inconsistencies during each translation step.

The problem of achieving portability for an application program across a set of environments centers on the program's interaction with each of its interfaces. This problem has two principal parts. First, a consistent structure must be maintained in each environment, as far as possible, for those elements of each interface that are actually used by the application. Second, the program must be adapted in each environment to deal with any remaining inconsistencies.

The developers of the system software for each environment and the designers of the application must jointly address the first part of the problem. System software designers can support portability by providing interfaces that are likely to be available in a similar form in many environments. Application designers can enhance the portability of their programs by using only the most portable interfaces and interface elements, and by isolating and documenting the program portions most likely to require adaptation.

Those who actually port the application to each new environment must face the second part of the problem — adaptation. Where a large amount of software is to be ported between the same environments, developing and using translation tools to assist with this process might be cost-effective.

We can now list some specific strategies that system designers or application programmers can use to help solve the problems outlined above. Each of these strategies represents a possible partial solution. Several might be needed in combination to provide satisfactory portability in a specific situation. The strategies are grouped into three categories:

(1) Strategies that maintain identical execution-time interfaces by porting system components that form the interface. This approach can reduce or eliminate the changes needed when subsequently porting applications. These strategies include the use of portable translators, portable libraries, and portable operating systems.

(2) Strategies that maintain identical or nearly identical interfaces for different system components by adhering to appropriate standards. Important strategies in this category include standard languages, standard libraries, and standard operating system interfaces.

(3) Strategies that assist in the adaptation of programs to a target environment. These include automatic translation, dynamic adaptation, and designing for portability.

I will further examine these strategies in the remainder of this article, beginning in the next section with strategies from groups (1) and (2) that concern languages and libraries. These strategies will be reviewed briefly, since they are relatively well known. Next, I will examine the strategies of portable operating systems and standard operating system interfaces in detail, and will conclude with brief consideration of some support strategies in group (3).

Languages, translators, and libraries

The first line of attack for achieving useful portability of applications has always been the use of a common programming language, one for which translators might be available in many target environments. The development of Fortran in the 1950s made this type of portability possible for the first time. Nevertheless, because of the subtle differences that can exist in implementations of supposedly identical languages, unexpected obstacles to portability remain. These problems have been reduced with the development of standard definitions for various languages. These definitions might be de facto standards established by common use, or they might be formal standards established by standardization bodies. The principal international and US organizations involved in computer-related standardization are identified in the sidebar. (Many other countries have similar national organizations.) Formal ISO standards now exist for Fortran, Cobol, Pascal, and Ada, among others; an ANSI standard was recently approved for C.

Even when a language is standardized, however, differences might be introduced by the behavior of different translators (compilers or interpreters). Translators might make different assumptions about issues not resolved by the standard, such as numeric precision. Compiler directives might have very different formats, and error-handling strategies might differ.

Some of these problems can be avoided through the use of portable translators. In this approach, the compiler or interpreter itself is ported to each target environment.

The "front end" of a portable translator, which processes the source code, is identical (at its own source level) for all environments. Thus, the same set of programs will be recognized in each environment, and each program will have the same interpretation.

Adaptation is clearly necessary in the "back end," which generates the machine instructions, since this process depends on the target architecture. This adaptation can be called *retargeting*. In the ideal case, however, each code sequence has an identical effect.

A disadvantage of this method is the need to establish a generic model for the CPU to provide an interface between the portable front end and the system-dependent back end. This can lead to inefficiencies if the generic model does not closely match the actual architecture.

Figure 5 illustrates the porting process using a portable translator. Both the applications and the translator must be ported. However, the translator is ported only once. Usually little or no adaptation of the application programs will be required. Examples of portable translators (all compilers) include UCSD Pascal, Turbo Pascal, and the Unix Portable C Compiler.

Usually, programs are ported either in executable form or in their source language representation. Occasionally, porting program modules in an intermediate form such as "object" representation is useful. In this case, portable linkers or standard object code formats can be useful to ease the portability problems. No examples of widely used portable linkers exist. An IEEE trial-use standard (designated 695-1985*) defines a common object code format (known as Microprocessor Universal Format for Object Modules or MUFOM). However, this format is not supported by many linkers or compilers. Revisions to this standard are being developed that can lead to more widespread use.

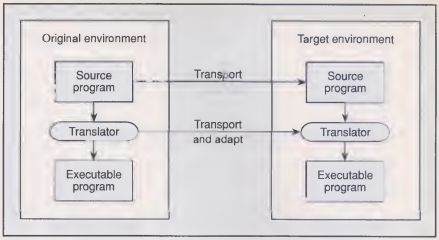


Figure 5. Porting a program using a portable translator.

Standard languages and portable compilers bridge the gap between programs and the variety of CPU interfaces that exist in target environments. However, many important higher level operations are typically omitted from programming-language

* IEEE standards can be obtained from the IEEE Service Center, PO Box 1331, Piscataway, NJ 08855-1331.

Principal computer-related standards organizations

International organizations

ISO: The International Organization for Standardization, a transnational body that develops and approves industry standards in a wide range of areas, including information processing, but excluding electrical and electronics engineering. The ISO is composed of member bodies from about 100 participating countries.

IEC: The International Electrotechnical Commission, the organization responsible for international standardization in the electrical and electronics fields. Like the ISO, the IEC is composed of member bodies from each participating country.

JTC-1: The Joint Technical Committee formed by the ISO and the IEC to coordinate standards work in information processing.

CCITT: The International Telegraph and Telephone Consultative Committee, a component of the United Nations International Telecommunications Union. The CCITT's function is to develop standards related to telecommunications to facilitate global compatibility of communication systems.

US organizations

ANSI: American National Standards Institute, a US organization that approves and manages standards proposed by "accredited standards committees" (ASCs), which carry out initiation and development of standards. ANSI also serves as the US member body in the ISO and the IEC.

X3: This is the principal accredited standards committee for ANSI in the field of information processing.

CBEMA: The Computer and Business Equipment Manufacturers Association, which serves as the secretariat for X3.

EIA: The Electronic Industries Association, an organization that produces standards useful in the electronics industry.

NIST: The National Institute for Standards and Technology, which develops and maintains standards for the US government, including Federal Information Processing Standards (FIPS). NIST was formerly known as the National Bureau of Standards (NBS).

IEEE: The Institute of Electrical and Electronics Engineers is an active developer of standards. The majority of its standards are in the area of information processing and have been developed within the IEEE Computer Society. The IEEE is an accredited standards committee for ANSI, comparable to X3. However, most of its standards are developed with international participation and have a unique transnational character as well.

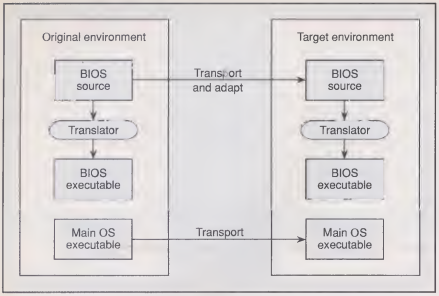


Figure 6. Porting CP/M and MS-DOS. (BIOS = basic input/output system)

definitions. These can include string handling, numeric algorithms, and even file and I/O processing. These additional needs are often addressed by procedures supplied through standard or portable libraries. Such libraries are implemented or transported into each environment, providing an extended set of services on which portable applications can rely. Examples include the common C function library; the Numerical Algorithms Group, or NAG library, for numerical algorithms; and various standard packages used in Ada environments.

Standard or portable languages, compilers, and libraries are important tools to use in the quest for portability. However, these mechanisms still define only part of the environment interface that many applications need. Elements such as file structures, device control, memory management, asynchronous event handling, or the user interface are not adequately defined by most language standards or library specifications. When requirements for communication, concurrency, or timing constraints exist, conventional languages are clearly insufficient. If the operating system is designed to support portability, it can play a key role in solving these additional problems.

Portable operating systems

One strategy to ensure consistency of interfaces between an application program and the operating system is to use the same operating system in all target environments. This can be achieved if the operating system itself is portable.

Early operating systems for large computers were developed and supplied by manufacturers as an integral part of the computing system. These suppliers had little interest in adapting their operating system to a different machine, or adapting someone else's OS to theirs. Moreover, many major applications were also supplied by the manufacturer, and users and programmers alike usually did all of their work in a single environment. Application portability was not an issue.

As computers became more widespread in the 1960s, however, operating systems from independent sources began to appear. Universities acquired computers and developed their own operating environments, such as CTSS (compatible time-shared system) and Multics (multiplexed information and computing service) at MIT. Minicomputers gained usage in many laboratories, where researchers developed system software to meet their own requirements.

The rapid progress of technology and architecture in the 1970s, and the continued proliferation of computers of many types, required many users to deal with a variety of computing environments. At the same time, software development costs increased rapidly. Both developers and users found it desirable to be able to use the same applications in each environment, and to rely on consistent user interfaces and system conventions. This need greatly increased the attractiveness of a portable operating system.

The programmers who first attempted to port an operating system, however, found the task hampered by two difficulties that do not generally arise in porting applications:

(1) The structure of an operating system is closely tied to the hardware structures it controls. In early systems, there was no effort to localize this dependence, so widespread adaptation was required.

(2) For efficient performance, plus direct access to hardware resources, most early operating systems were written in assembly language.

Later developments helped to alleviate these key problems. Languages and compilers have emerged that are more suited to implementation of system software, including operating systems. Important examples include BLISS, PL/M, BCPL, C, and Ada. These languages are designed to support efficient implementation and better control of hardware resources, yet remain (more or less) portable to different hardware environments. Collectively, they are sometimes called *system implementation languages* (SILs). An increasing proportion of contemporary operating systems are now implemented using SILs.

A second important development has been the emergence of target classes of environments that share the same processor architecture. Families of computers developed by a single manufacturer (such as the IBM System 360-370 and the Digital VAX) are important examples of such classes. The age of the microprocessor has also given rise to computer classes with the same CPU but different memory and I/O structures.

Computer families from a single manufacturer typically have highly consistent interfaces throughout and are supplied with operating systems with binary portability (such as MVS for 360-370 architectures and VMS for the VAX). The similarity of microprocessors with a common CPU has enabled some operating systems to be reasonably portable within such a class in spite of other hardware differences.

The CP/M and MS-DOS microcomputer operating systems have been ported to a variety of microcomputer systems using the Intel 8080 and 8086 architectures, respectively. The environments targeted by these systems differ primarily in the structure of their I/O devices. To accommodate these differences, the systems support a

hybrid approach to porting, as illustrated in Figure 6.

Device drivers are localized in one module that is supplied in source form for adaptation. This module is commonly called the BIOS, for basic I/O system. The remaining system modules are ported in binary form. The simple memory structures in these systems enable the necessary linkages to be easily established. Typically the BIOS is permanently resident, and its routines are accessed through a transfer table.

Operating system portability is much more difficult across classes with different processor architectures, but this has been achieved in some notable cases. CP/M offers a limited example; although designed for the Intel 8080, it has been adapted to environments using other processors, including the Intel 8086 and Motorola 68000. These versions offer consistent user interfaces and file structures, but make no claim to support direct portability of applications. In most examples, though, the goal of porting an operating system to targets with varying architecture is to more effectively support source portability for applications (binary portability is clearly not possible).

A number of experiments in the late 1970s and early 1980s were directed at transporting operating systems to new environments (including different processors) and at creating new operating systems designed to be portable.

A few operating systems were successfully ported even though portability was not a goal of their original design. Two important examples are Brinch Hansen's Solo[2] and Stoy and Strachey's OS/6.[3] The careful design strategies followed by these operating systems minimized the obstacles cited above, making porting much more feasible. First, they provided simple, modular designs with localized hardware dependencies; second, most of their code was written in system implementation languages.

Of course, portability can be most effectively achieved if it is a goal of the original system design. Such was the case with MUSS,[4] Thoth,[5] TRIPOS,[6] and Pick.[7] The first three of these were research systems, but TRIPOS later formed the basis for Amiga DOS, the operating system for the Commodore Amiga. Pick, a business-oriented OS built around a relational database, is distributed in a number of commercial environments.

Portability was not an important concern with the earliest versions of Unix, but it became a key objective of later versions.[8] Today, Unix is the most extensive example of operating system portability; versions of this system have been implemented for almost any environment with the resources to support it.

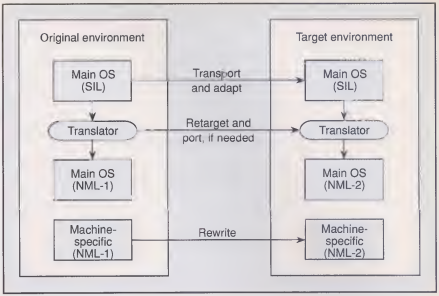


Figure 7. Porting an operating system: conventional approach. (SIL = system implementation language; NML = native machine language)

Portability is likewise a fundamental goal of Mach, the distributed version of Unix developed at Carnegie Mellon University.[9] This operating system has been adapted to a number of environments ranging from multiple VAX configurations to Next workstations.

Each of these operating systems was written in a system implementation language. TRIPOS was written in BCPL, an ancestor of C. Thoth used a variation of BCPL, and Unix was written in C. MUSS used several special-purpose languages. Pick is written in an assembly language designed for a generic machine architecture, similar to the approach taken by some portable translators. The Pick assembly language representation is portable because the generic machine language is translated to the native machine language of each target environment during the porting process.

The use of SILs reduces the problem of adapting to the hardware, but does not eliminate it. Porting an operating system typically requires adaptation of a language translator to produce code for the target environment. This translator can initially operate in the original environment; it can then be ported to the new environment like an ordinary application after the operating system port is complete.

A substantial portion of the operating system code can be designed to be independent of the actual hardware architecture, provided the architectures in the target class do not vary too radically. This code has a high degree of source portability, and can be ported in a straightforward manner once a translator for the target environment is available. However, system-specific procedures are normally needed to control I/O devices and specialized CPU resources. Due to the need for high performance and precise hardware control, some of this system-specific code is usually written in assembly language. The system-specific code must be redesigned and reimplemented for each new environment.

The type of operating system portability described above is found in systems such as TRIPOS and Unix. Figure 7 illustrates the process of porting such an operating system. This figure depicts the source-level porting of the main body of the system code, the optional retargeting and porting of the language translator, and the reimplementation of the machine-specific portions. The acronym NML in this figure stands for native machine language.

MUSS and Thoth use a somewhat similar structure, but there is an important difference. The body of these operating systems, written in a SIL, is compiled into machine language for a generic machine architecture. This machine language is then loaded into the target environment, where it is interpreted by an emulator on the target CPU. The emulator must be reimplemented for each environment, but the main part of the operating system, in principle, could

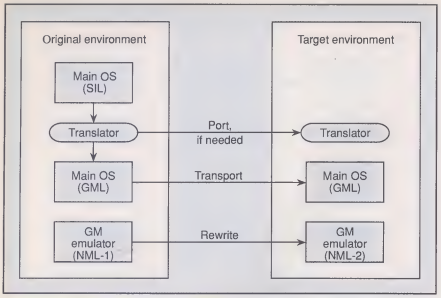


Figure 8. Porting an operating system: generic machine approach. (SIL = system implementation language; GML = generic machine language; NML = native ma-

be ported in its generic machine language form.

Figure 8 illustrates the potentially simplified porting process for this type of operating system. In this figure, GML stands for generic machine language. In principle, the translator need not be ported to port the OS. In practice, though, it is usually required for translating other programs in the new environment. In any case, the translator does not have to be retargeted, since the generic machine language does not change.

Use of a generic machine approach can be beneficial, but the characteristics of this machine (instruction set, hardware resources, etc.) must be chosen carefully. They must map in a reasonable way to each of the potential target environments. If they are too low level, they cannot be efficiently emulated. If they are too high level, a large portion of the operating system will require reimplementation for each environment. Theaker and Frank[4] and Cheriton[5] discuss these issues.

A variation of these strategies is used in porting the Pick system. Pick is designed in terms of a generic machine, but the machine code is compiled rather than interpreted. Thus, the porting process is similar to Figure 7. The Pick SIL is an assembly language for the generic machine. Porting Pick requires writing an assembler for this language that generates machine code for the target environment. Cook and Brandon[7] provide a more detailed discussion.

The most widespread porting of an operating system to a variety of environments has occurred for the Unix operating system. Although early versions of Unix were written in assembly language, all later versions are written almost entirely in C. The C language was designed to implement Unix, but has since been adapted for general use in a great number of environments. The widespread availability of C compilers greatly enhances the portability of this operating system. Unix portability has been the subject of several papers; Johnson and Ritchie[8] discuss an early port.

These examples show that porting an operating system can sometimes be achieved and is often worthwhile. Many newer operating systems are being designed for portability, and designers of future ones will do well to follow this example. However, experience so far has shown that porting an operating system is not an easy task. Moreover, a large collection of system tools and utilities might need to be ported along with the OS to create a complete and useful environment. According to the references cited in this section, the effort required to reimplement an operating system plus translators and essential support software in a new environment ranges from several man-months to several man-years.

In many situations, therefore, porting the operating system is not feasible. Even if this were not true, an OS that is used effectively in one environment might not be suitable or desired in another. The OS strongly defines many of the characteristics and limitations of a total environment, and no OS is suitable to all needs. If the OS is not to be ported, the strategy in the next section can be considered.

Standard operating system interfaces

The rapid spread of Unix has suggested to some observers that the problems of portability will be solved in the near future by the emergence of a universal operating system, or at least a small number of such systems. Indeed, a handful of operating systems seem to dominate the computer field. However, none of these systems has been found satisfactory for all purposes, and many new operating systems continue to be developed. The "Catalog of Operating Systems" in the text by Lane and Mooney[10] identifies more than 500 operating systems as of early 1988. A large portion of these was developed within the last decade. Dozens more have appeared since the catalog was prepared. Many new systems are motivated by new requirements and unsolved problems in distributed systems, real-time systems, user interface methods, etc. A universal operating system does not appear to be on the horizon.

Many versions of Unix, in fact, differ widely in their internal structure. However, one thread that binds them is a common program interface in the form of a collection of system calls for use by application programs. Moreover, many additional operating systems not considered to be versions of Unix adhere as far as possible to the Unix program interface. This concept can provide good support for source portability even though the operating systems themselves are quite different.

The principal objective of a common or standard operating system/program interface is to provide a uniform source-level model that applications can use for access to system services that are not defined directly by programming languages or standard libraries. The services that fall into this category depend on the language and library, but they can include services for memory management, timing, exception handling, process creation and control, communication, and others. Many applications require none of these services, but others require many of them. It is common for a group of operating systems to provide functionally equivalent support for many services of this type, but to present them in arbitrarily different ways, creating barriers to portability. Standard interfaces between operating systems and application programs offer a strategy to overcome this problem.

The interface discussed in this section is the direct interface between an application program and an operating system, as shown

in Figure 3. This interface, in turn, supports a variety of indirect interfaces. It is sometimes designated the OSI, for operating system interface, but this abbreviation has other meanings which can lead to confusion (in particular, the well-known ISO/OSI communications model). The designation API, for application program interface, has also been used. However, Figure 3 illustrates that there are several interfaces that could fit this description, even if we restrict attention to the direct ones. To be most descriptive and avoid confusing abbreviations, we will refer to the interface of concern as the OSAPI (for operating system/application program interface).

A standard OSAPI is normally defined in two parts. First, a functional definition specifies the covered services in terms of the required semantics of each system call, including input and output parameters. This definition does not specify how the operations are expressed in a language, or how they are represented in a machine architecture. Second, a set of language bindings provides a concrete form for expressing the functions in suitable languages. This expression typically has the form of a collection of procedures; thus, the programmer views the OSAPI as a standard procedure library.

To enable portability, the OSAPI must be mapped onto the services provided by the actual operating system in each environment. This can be done for any OS that can provide the necessary functionality. Overhead is minimized if the OS includes the standard OSAPI in its native program interface. However, in many cases, the OSAPI can be satisfactorily implemented as a separate software layer. If the OS itself lacks some of the required functionality, this can sometimes be added in the interface layer.

A standard OSAPI is focused on providing the necessary support for portable applications. It is not generally concerned with services that cannot be easily used in a portable way. For this reason, the OSAPI specifies only a subset of the services provided by most target operating systems. Figure 9 illustrates the use of a common OSAPI to enable portability. The porting procedure is quite similar to the general method shown in Figure 2a; however, the OSAPI reduces the adaptation required.

Both operating system developers and industry-wide standards groups have begun to recognize the value of such interfaces. This is leading to the emergence of formal standards, as well as standards developed by single companies or industry groups.

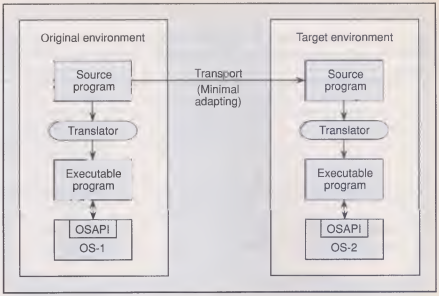


Figure 9. Use of a common operating system/application program interface (OSAPI).

Several important standards and standards development projects are described in the remainder of this section.

Posix. Despite the rapid spread of Unix into many environments, the potential portability of applications throughout these environments has not been fully realized due to subtle differences in the many Unix versions. This problem persists in spite of similar system calls in each version, because some differences continue to exist in the system calls as well.

Posix is the popular name for IEEE standard 1003.1-1988 (Portable Operating System Interface for Computer Environments) developed to address this problem by identifying and defining a standard OSAPI based on the Unix system calls. There has been a great deal of interest in this standard, which is expected to enable source-level application portability among all operating systems with a Unix-like program interface. Posix has also been adopted as a Federal Information Processing Standard (FIPS) and approved by ANSI and ISO.

The Posix project is considering standardization of other elements of a Unix-like environment, including user interfaces and system tools, and studying additional related issues such as security and real-time support. In the following discussion, the term Posix refers only to the base standard for the system call interface.

The body of the Posix standard specifies a collection of system calls in a number of functional areas. Table 1 summarizes these areas. The calls are defined directly in the syntax of the C language, so that Posix has a built-in C binding. No other language bindings are currently defined, but an Ada binding is under study. Functions that are considered to be handled by the standard C Library are not specified as Posix system calls; in particular, there are no system calls for memory allocation.

In addition to system calls, a detailed model of terminal handling is defined. There are also specifications for password encoding and for a data interchange format. Some elements of the typical Unix interface are omitted. There are no system management functions, and there is no concept of a "super-user."

Posix provides a detailed definition for a fairly specific type of environment. As such, it limits its class of potential environments, but the number of systems within that class is large. For suitable targets, Posix supports a high degree of portability. The Posix standard is new, and few Unix-like systems fully conform to it; but the great majority of such systems are expected to conform in the future.

MOSI. While the Posix standard was evolving for Unix-like systems, another IEEE group was studying OSAPIs suitable for smaller environments. This effort led to the recently approved IEEE standard for Microprocessor Operating System Interface (855-1990), commonly known as MOSI. MOSI has attracted far less attention than Posix, but it offers a solution to a complementary problem: supporting portability as far as possible over a range of systems with widely divergent structure.

Table 1. Comparison of functional areas addressed by Posix (Portable Operating System Interface for Computer Environments), MOSI (Microprocessor Operating System Interface), and CTRON (part of The Real-time Operating System Nucleus).

	Posix	MOSI	CTRON
Process Management	Create and terminate Get information	Create and terminate Get and set information Suspend and resume	Create and terminate Get and set information Suspend and resume Real-time scheduling
Program Management	Load and execute	Load into new process Simple overlays	Load and execute
Interprocess Communication	Signals Pipes	Event flags Mailboxes Semaphores	Event flags Mailboxes Semaphores and locks Rendezvous
Memory Management	None (relies on C Library)	Heap allocation Memory pools	Heap allocation Memory pools Virtual storage Swapping control
File Management	Create and delete Open Synchronous access Get and set information Directories	Create and delete Connect and open Synchronous access Asynchronous access Get and set information Directories Lock and replicate	Create and delete Connect and open Synchronous access Asynchronous access Get and set information Directories Indexed files Journaling
Device Management	Generic (as files) Terminals	Generic (low level)	Generic (basic) Device classes
Time Management	Delay Alarms	Delay Private timers Time of day	Delay Private timers Time of day
Event Management	Signals	Exceptions	Exceptions Interrupts
Environment Access	System characteristics Current directory Process and group IDs Strings and variables	System characteristics Current directory Command strings Standard devices	Current directory
Other	Password encryption System data access	Transactions	Transactions System management Database access Communication and switching

Whereas Posix is a detailed and complete specification for a narrow class of target environments, MOSI is less detailed and less complete, but targets a broader class of systems. The MOSI standard is directed primarily at microsystems, but it can be useful in larger environments as well.

MOSI consists primarily of a collection of system calls and a set of language bindings. The language bindings are suggested, not officially included in the standard. The system calls are presented in a descriptive form that is not bound to the syntax of a specific language. Table 1 also lists the categories addressed by these system calls.

The appendixes to the MOSI draft standard provide suggested language bindings for Pascal, C, Fortran, Cobol, Ada, and PL/M. Each of these bindings attempts to view the MOSI functions in terms suitable to programmers in that specific language. In some cases, functions are consolidated, divided, or even omitted, and parameter lists can be changed. Memory management or file access operations defined in some languages are not duplicated. There is no attempt to add dynamic memory allocation to static languages such as Fortran or Cobol (but process management can be provided for these languages).

Experimental implementations of MOSI have shown its potential usefulness. One such implementation has demonstrated the addition of MOSI-defined functionality to a weaker operating system, namely CP/M. To date, the standard has seen little commercial use. However, some of the MOSI concepts have been adopted in the definition of the CTRON interface, part of the TRON project described next.

The TRON project. A consortium of companies based in Japan has organized an ambitious undertaking called the TRON project[11] (for The Real-time Operating System Nucleus) to develop a collection of standards suitable for use in a next-generation distributed network environment. The envisioned network would include local connectivity of processors embedded in "intelligent objects" along with conventional workstations, servers, and host processors. The project is based on the vision of Ken Sakamura of the University of Tokyo and is directed by the TRON Association, an industry organization with about 150 members. Most TRON Association member companies are Japanese; however, membership is open to all companies, and many major Western companies are represented.

The principal subprojects of TRON include definition of a CPU architecture and three types of operating systems. Each of these efforts is already leading to commercial products; other subprojects are in earlier stages of development.

The three TRON operating system projects are designated ITRON, BTRON, and CTRON. ITRON is designed for use in embedded systems, and BTRON is intended for workstations. CTRON, the largest system, is intended for network hosts and servers.

Each of these projects defines an OS API that can be ported to suitable hardware environments. The target environments for ITRON and BTRON are microprocessor systems, possibly but not necessarily using the TRON CPU. ITRON defines a tight set of functions directed at real-time multitasking, including scheduling control and several mechanisms for interprocess communication. The BTRON interface is focused on sophisticated user interfaces and file systems, including integral support for multiple languages.

CTRON is intended for use on a variety of larger system environments. It is by far the largest specification, with over 100 functions defined in the kernel alone, and hundreds more in various extended operating system modules. As is necessary for such a large specification, a collection of acceptable subsets is also defined.

CTRON is oriented especially toward real-time and switching requirements. Among the notable functions found in the kernel are cyclic task scheduling, interrupt handler control, and optional virtual memory management. The I/O module provides both generic device handling and functions for several specific device classes. Extended operating system modules are defined for communications, database access, file management, program management, and switching control. The communications module defines implementations of several layers of the ISO/OSI communication model. The database model is based on the standard query language SQL. Much of the CTRON interface is comparable to those defined by Posix and MOSI. Table 1 also summarizes this interface.

Each of the TRON operating systems interfaces and the TRON CPU specification have been used in a number of implementations. They are providing consistent interfaces in spite of significant differences in the underlying systems. The extent of industry support for TRON, not only in Japan, ensures that these standards will play a role in future system design.

The TSG-1 study. Many formal standards have been established to improve the consistency of application program interfaces. However, some important areas have no suitable standards, while others have several competing standards. A technical study group, designated TSG-1, has been formed within JTC-1, the Joint Technical Committee formed by ISO and IEC to coordinate international information processing standards. The charter of TSG-1 is to study the problem of application portability. This group hopes to develop a model for application portability similar to the OSI model for communications that can serve as a unifying basis for standards development in the future. The work is planned for completion in 1991.

IBM's System Application Architecture. Individual companies are also addressing the problems of consistency and portability within their product lines. This was the motivation for IBM's System Application Architecture (SAA),[12] which is intended to provide consistency across a range of systems from large mainframes running MVS or VM, through midsized systems, to small PS/2s running OS/2.

SAA has three principal elements:

(1) A set of rules for consistent user interface design (within the limits of terminal capabilities);
(2) A collection of protocols for communication; and
(3) The Common Programmer Interface.

CPI is designed to provide programmer support for application portability and so is most relevant to this discussion. Unlike the interfaces considered above, the CPI interface defines a high-level collection of languages and complete services (such as those for database and communications). These tools themselves supply the programmer's interface, so consistency at lower levels is not required.

Other projects. Other industry and government organizations also have active projects to develop standards aimed at supporting application portability. Two examples are the Open Software Foundation and the US Navy's Next Generation Computer System project.

Several computer system developers are giving increased attention to the problem of supporting portability, at least across different systems from the same company. Besides IBM's high-level SAA described above, a portable subset of available system calls has been defined for IBM's OS/2 and PC-DOS. Programs that use only this subset, called "Family API," are assured of (binary) portability across present and future versions of either of these operating systems.

Adapting applications to new environments

The last class of support strategies includes those that aid the process of adapting programs (at the source level) to environment differences that cannot be removed or hidden by common interfaces. To some degree, this is a manual chore. However, if some straightforward transformations must be made to a large volume of software, it might be worthwhile to consider the construction of a specialized conversion program.

Clearly, revisions will be easiest for programs designed with this problem in mind. A standard programming language should be used, and nonstandard extensions of the language should be avoided. Standard or portable procedures should be used for additional required functions wherever possible. A good modular structure that

isolates all code likely to be nonportable will greatly ease the job of the program adapter.

A final strategy that is often overlooked is dynamic adaptation. Standard interfaces like MOSI and Posix include methods for programs to determine during execution the characteristics of the system on which they are running. This information can be used to select the most appropriate methods for activities that might be system dependent. Unix programs are often designed to behave quite differently when invoked interactively than they do when invoked from a command script. The same principle can be applied to many portability problems.

Further issues

This section provides a brief mention of a few related issues not discussed elsewhere in the article.

User interfaces. It is often not desirable that the user interface provided by an application in one environment be duplicated in other environments. Instead, a consistent user interface might be desired in a specific environment, for all applications. A major portion of the problem of adapting an application during a port might concern the user interface. Mechanisms that define a more consistent programmer's view of user interface objects and activities are needed, even when the actual interface is quite different (for example, Unix versus Macintosh). The user interface component of SAA and several standard windowing environments are important steps in this direction.

Dynamic portability in a network. This discussion has assumed that porting a program is an off-line activity. In a network environment, however, porting programs rapidly on demand might be desired. To achieve this capability among dissimilar systems, any required adaptation must be fully automated.

Internationalization. There is growing interest in better international exchange of programs, sometimes via networks. Such programs must be adapted to the language and cultural conventions of their new environment. Obvious problems like message translation can be handled by careful isolation, as in many Macintosh programs. Less obvious problems can include differences in the interpretation of symbols, or in suitable representations for money or dates. A common interface for international differences will be an important future project. This concern is receiving particular attention in the TSG-1 study.

The need for improved solutions for application portability is becoming recognized as a critical software engineering problem. However, many designers might be unaware of the wide range of old and new strategies available to meet this need. A sizeable body of experience is available, a number of techniques have proved effective in particular cases, and a variety of standards have emerged to help both application and system designers address this problem. Through this article, I have tried to offer a systematic view of some of these resources. ■

Acknowledgments

The ideas expressed in this article were developed through a variety of research and teaching activities related to operating systems and software portability. I thank the IEEE Microprocessor Standards Committee for partial support of my work on the MOSI standard and the TRON Association for assisting my research on the TRON project. I am also grateful to West Virginia University, and especially to the Department of Statistics and Computer Science under the leadership of Donald Butcher, for continuing support of my research, teaching, and writing activities. Finally, I thank the anonymous referees for their careful comments, which led to some significant improvements in this article.

This work was supported in part by a grant from Nippon Telegraph and Telephone.

References

1. *Software Portability*, P.J. Brown, ed., Cambridge University Press, Cambridge, England, 1977.
2. M.S. Powell, "Experience of Transporting and Using the Solo Operating System," *Software — Practice and Experience*, Vol. 9, No. 7, July 1979, pp. 561-569.
3. C.R. Snow, "An Exercise in the Transportation of an Operating System," *Software — Practice and Experience*, Vol. 8, No. 1, Jan.-Feb. 1978, pp. 41-50.
4. C.J. Theaker and G.R. Frank, "MUSS — A Portable Operating System," *Software — Practice and Experience*, Vol. 9, No. 8, Aug. 1979, pp. 633-643.
5. D.R. Cheriton, "Thoth, a Portable Real-Time Operating System," *Comm. ACM*, Vol. 22, No. 2, Feb. 1979, pp. 105-115.
6. M. Richards, "TRIPOS — A Portable Operating System for Minicomputers," *Software — Practice and Experience*, Vol. 9, No. 7, July 1979, pp. 513-526.
7. R. Cook and J. Brandon, "The Pick Operating System, Part 2: System Control," *Byte*, Vol. 9, No. 12, Nov. 1984, pp. 132-133, 474-483.
8. S.C. Johnson and D.M. Ritchie, "Portability of C Programs and the Unix System," *Bell Systems Tech. J.*, Vol. 57, No. 6, Part II, July-Aug. 1978, pp. 2,021-2,048.
9. R. Rashid, "Machine-Independent Virtual Memory Management for Paged Uniprocessor and Multiprocessor Architectures," *IEEE Trans. Computers*, Vol. 37, No. 8, Aug. 1988, pp. 896-907.
10. M. Lane and J. Mooney, *A Practical Approach to Operating Systems*, PWS-Kent, Boston, Mass., 1989.
11. *IEEE Micro*, TRON project special issue, Apr. 1987.
12. D.E. Wolford, "Application Enabling in SAA," *IBM Systems J.*, Vol. 27, No. 3, 1988, pp. 301-305.

James D. Mooney is an associate professor in the Department of Statistics and Computer Science at West Virginia University and has taught courses in operating system design and computer architecture since 1979. With Malcolm Lane, he coauthored a text entitled *A Practical Approach to Operating Systems*. He has been an active member of IEEE Standards Working Group 855 (MOSI) and its chair since 1985; a consultant for the TRON project since 1988; and a contributor to the TSG-1 study group on application portability.

Mooney received the BS degree from the University of Notre Dame, and the MSc and PhD degrees from Ohio State University, all in electrical engineering. He is a member of the IEEE, the IEEE Computer Society, ACM, Tau Beta Pi, Eta Kappa Nu, and Upsilon Pi Epsilon.

To contact the author or obtain a more complete bibliography on portability issues, write to James D. Mooney, Department of Statistics and Computer Science, West Virginia University, Morgantown, WV 26506. His electronic mailing address is jdm@a.cs.wvu.wvnet.edu.

Garnet

Comprehensive Support for Graphical, Highly Interactive User Interfaces

Brad A. Myers, Dario A. Giuse, Roger B. Dannenberg, Brad Vander Zanden, David S. Kosbie, Edward Pervin, Andrew Mickish, and Philippe Marchal
Carnegie Mellon University

User interface software is difficult and expensive to implement.[1] Highly interactive interfaces are among the hardest to create, since they must handle at least two asynchronous input devices (such as a mouse and keyboard), real-time feedback, multiple windows, and elaborate, dynamic graphics.

The Garnet research project is creating a set of tools to aid the design and implementation of highly interactive, graphical, direct manipulation user interfaces. Garnet also helps designers rapidly prototype different interfaces and explore various user interface metaphors during early product design.

Most graphical interfaces are created using toolkits, collections of interaction techniques (sometimes called "widgets" or "gadgets") such as menus, scroll bars, and buttons. Examples include the Macintosh Toolbox and Xtk for the X Window System. Unfortunately, these toolkits are often difficult to use, since they contain literally hundreds of procedures. Also, many toolkits do not help the programmer create the most important part of the application — the graphics that appear in the main application window. In particular, the application must handle all input events (expressed at a low level as "the left mouse button went down," "the mouse is at (30,345)," etc.), deciding which operation to perform on objects and drawing objects with the underlying graphics package (which usually supplies operations such as "draw-line" and "draw-circle"). Furthermore, modifying or creating toolkit items is usually difficult or impossible.

Garnet helps create highly interactive user interfaces by emphasizing easy specification of object behavior, often by demonstration and without programming.

Higher level tools such as interface builders and user interface management systems[1] have not adequately addressed these problems. A conventional interface builder lets a designer graphically place user interface components in a window, thereby creating menus, palettes, and dialogue boxes. Examples include Next's Interface Builder, Smethers Barnes' Prototyper for the Macintosh, and UIMX for X Windows. These programs let a designer place only preprogrammed interaction techniques in windows, and they usually allow only a few parameters to be set. Often, a designer will type the name of a procedure to be called when the interaction technique is executed. Designers cannot modify or design the interaction techniques themselves, and the interface builders do not address application-specific graphics at all.

A user interface management system helps a designer handle the dialogue, or sequencing, aspects of the user interface — that is, what happens after each user action. Such systems have generally not addressed the creation of toolkit components (menus, scroll bars, etc.) or the specification or management of application-specific graphical objects (the contents of application windows).

A number of features differentiate Garnet from other user interface tools, including an emphasis on handling objects' runtime behavior (how they change when the user operates on them) and on handling all visual aspects of a program's user interface, including its graphics and the contents of all application-specific windows.

For example, when creating an application that lets the user manipulate boxes connected by arrows (such as a graph edi-

0018-9162/90/1100-0071$01.00 © 1990 IEEE

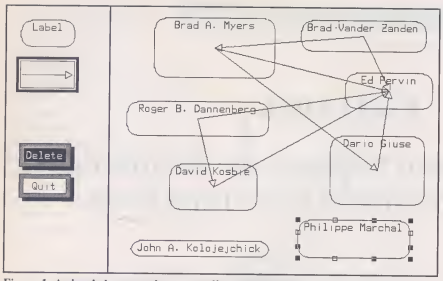


Figure 1. A simple boxes-and-arrows editor created with Garnet. An arbitrary number of new boxes and arrows can be added, and their initial position is specified using the mouse. Any existing box or arrow can also be selected, changed in position or size, and deleted. This entire editor was implemented in about three hours using the Garnet Toolkit.

tor or project-planning chart — see Figure 1), Garnet's user interface builder lets the designer draw the boxes and arrows and demonstrate how they should respond to the mouse. Menus, palettes, and dialogue boxes can be generated automatically or added graphically. Toolkits and interface builders in other systems might help build the menus and palettes, but they don't let the designer implement and manage the boxes and arrows themselves. Designers using those programs must code these elements directly using the underlying graphics package and input device handlers.

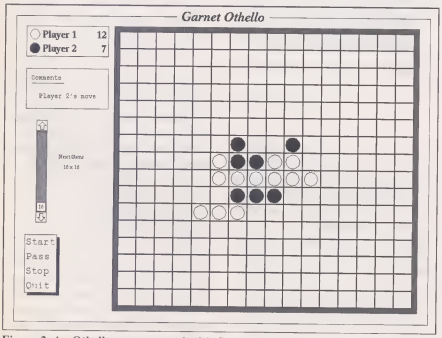


Figure 2. An Othello game created with Garnet.

Typically, coding application-specific graphics requires 10-100 times more effort than dealing with menus and buttons, so Garnet should save programmers significant time.

This article provides an overview of all parts of Garnet, which stands for "generating an amalgam of real-time novel editors and toolkits." Previous articles[2-5] have concentrated on individual aspects of the system. (There is also a complete reference manual for the Garnet Toolkit.[6])

Garnet is entirely "look-and-feel independent," which means the designer can either create user interfaces with an original graphical appearance and behavior or choose from a set of predefined appearances and behaviors.

The system contains both low-level and high-level tools. The low-level tools, called the Garnet Toolkit, use a number of mechanisms to help implement user interfaces, including a prototype-instance object system, a constraint system (which allows relationships among objects to be declared once and then maintained by the system), automatic graphical object updating (so that the system refreshes the display when objects change), and separate specification of input handling from graphics.

Garnet's high-level tools include the Lapidary interface builder; the Jade dialogue box creation system, which automatically creates menus and dialogue boxes from a list of their contents; and the C32 spreadsheet for specifying complex constraints. Garnet is implemented in Common Lisp and uses the X Window System through the standard CLX interface from Common Lisp to X Windows. A version for the Macintosh is also being developed. Garnet is therefore portable and runs (so far) on CMU, Lucid, Allegro, Harlequin, and TI Common Lisps and on many hardware platforms. Garnet does not use the Common Lisp Object System (CLOS) or any Lisp or X Windows toolkit (such as Interviews,[7] CLUE, CLIM, or Xtk).

Coverage

Garnet is designed to handle user interfaces that let users operate on graphic objects with the mouse and keyboard. Since the objects often represent application data, and since changes made on the screen to the graphics are translated into changes to the data, the user has the feeling he or she is directly manipulating the data. Similarly, the screen reflects changes the application makes to the data, possibly in response

to external events.

Garnet is suitable for

- box and arrow diagram editors (see Figure 1), like MacProject;
- conventional drawing programs, such as MacDraw;
- icon manipulation programs, like the Macintosh Finder;
- graphical programming languages in which computer programs are constructed using icons and other pictures, such as a flowchart;
- tree and graph editing programs, including semantic networks, neural networks, and state transition diagrams;
- board games, such as chess or Othello (see Figure 2);
- simulation and process monitoring programs, in which the user interface shows the status of the monitored simulation or process and lets the user manipulate it;
- user interface construction tools (we implemented Garnet using itself); and
- some forms of CAD/CAM programs.

Garnet does not have a significant text editing component, but it does provide small editable strings that might be used as labels or fields in a table or dialogue box.

Garnet's primary influence is Peridot,[8] a construction tool that lets users create toolkit items without programming. Peridot lets nonprogrammers create many types of interaction techniques, including most kinds of menus, property sheets, buttons, scroll bars, percent-done progress indicators, sliders, and iconic and title line window controls. Like Garnet, Peridot uses constraints. However, Peridot lacks a programming interface and offers no way to use existing toolkit items or create application-specific graphic objects. (Many systems, including Thinglab[9] and Apogee,[10] have used constraints; see the sidebar.)

Garnet's Jade dialogue editor, which automatically constructs dialogue boxes from high-level specifications, was influenced by the Interactive Transaction System.[11]

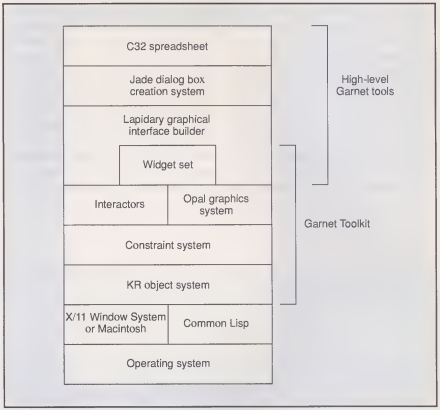


Figure 3. The structure of the Garnet system.

The Garnet Toolkit

Garnet contains a number of different components grouped into two layers. The Garnet Toolkit (the lower layer) supplies the object-oriented graphics system and constraints, a set of techniques for specifying the objects' interactive behavior in response to the input devices, and a collection of interaction techniques. Designers using this layer must write procedures. The higher layer contains tools that let designers draw pictures to show how the interface should look and behave and to define parts of the interface at a high level. These tools automatically create code based on the user's specifications.

The toolkit itself is divided into several components (see Figure 3):

- an object-oriented programming system,
- a constraint system,
- a graphical object system,
- a system for handling input, and
- a collection of gadgets or widgets.

Using the X toolkit's terminology, the first four parts of the Garnet Toolkit are intrinsics (the mechanisms supporting the implementation) and the fifth is the widget set (a collection of menus, scroll bars, etc., with a prespecified look and feel).

Object-oriented programming system. Garnet's object-oriented programming system, called KR (for "knowledge representation"), supports a prototype-instance model for objects rather than the conventional class-instance model used by Smalltalk and C++. A prototype-instance model does not distinguish between instances and classes; any instance can be a "prototype" for other instances. All data and methods are stored in "slots" (sometimes called fields or instance variables). Slots that are not overridden by a particular instance inherit their values from their prototypes. Actually, KR does not distinguish between data and method slots. Any slot can hold any type of value, and in Common Lisp a function is just a type of value. Slot names start with colons and can contain any number of printing characters (for example, :left, :interim-selected, :obj-over).

An instance can also add any number of new slots. Unlike conventional class-instance models, this means that the number of slots in each object is highly variable — each object can have a different number of local slots, depending on which properties it wants to accept as defaults and which it wants to override. In fact, the number of slots can change dynamically. Slots can be explicitly removed from objects at any time. If a program sets the value of a slot that does not exist, then the slot is created automatically. The advantage of this fea-

ture is that it is easy to create slots to hold local data. For example, if a rectangle's color represents an application temperature, the application can create an instance of a rectangle and a slot in it called :temperature. A disadvantage of this flexibility is that compile-time or even runtime checking of slot accessing and setting is impossible, since all slot names are legal. Also, since slots can hold any type of value, there is no type checking at the KR level.

As an example of inheritance, assume that a top-level rectangle prototype has slots containing values for the rectangle's left, top, width, height, and color (see Figure 4). An instance using that rectangle as a prototype will typically — but not necessarily — override some of the values. A programmer could create an instance with a particular color and size, and then create more instances of that instance (see Figure 4). If a prototype's slot value changes, all instances of that prototype that do not override that slot immediately inherit the new value. Similarly, if an instance's slot is removed, the corresponding slot of its prototype will be used instead, if it exists. All objects (prototypes and instances) can be displayed on the screen.

The methods that implement messages sent to the objects can also change dynamically, which is not possible in conventional object systems like Smalltalk. In Garnet, the designer need only assign a new procedure to the appropriate slot; the new method will be used subsequently.

The prototype-instance model is more dynamic and flexible than the familiar class-instance model. A high-level tool such as

A constraints primer

Brad Vander Zanden and Brad A. Myers

Any large, complex application contains thousands of interdependent relationships. For example, a graphical application must deal with the relationships arising from laying out objects, displaying feedback for input operations, and keeping views consistent with the underlying data they represent. These relationships might include keeping text labels centered in boxes, making feedback objects follow the cursor during a move operation, and setting the color of an airplane icon according to whether a flight is early, on time, or delayed.

Constraints provide a convenient way to specify relationships and have them automatically maintained at runtime by a constraint solver. In contrast, a conventional programming language requires the application to both specify the relationships and do all the bookkeeping to maintain them. For example, suppose that an arrow must stay connected to the center of a box and that a label on the arrow must stay centered on the arrow. In constraint programming, whenever the application changes the box's position or size, the constraint solver automatically repositions the arrow and its label. In contrast, a conventional language forces the designer to write code to reposition the arrow and the label. This might not seem too onerous a task, but when an application contains thousands of such relationships, the bookkeeping needed to maintain them increases so rapidly that adding new functionality to the application becomes difficult, and the time required to debug the changes also increases substantially.

In addition to simplifying the creation of an application and increasing its robustness, constraints lend themselves to incremental recomputation. When a user changes one or more parameters in an application, or adds or deletes a number of constraints, most of the existing constraints remain satisfied and only a small number must be reevaluated. An incremental constraint-solving algorithm can automatically identify which constraints must be reevaluated and limit its solving to these constraints. Such an algorithm can be used with any application written for that constraint system. In contrast, a conventional language requires the designer to create a new incremental algorithm for each application. Of course, a conventional language also lets the designer take advantage of any special characteristics of the application, so that he or she can write a custom algorithm that might be faster than a general-purpose constraint solver.

Types of constraints

Constraints can be either one-way or multiway. One-way constraints let the constraint solver change only one object in the constraint to satisfy it; multiway constraints allow any object to change. For example, the arrow in Figure 5 (in the main text) is connected to the circle and the box by one-way constraints. If either the box or the circle moves, the arrow also moves to resatisfy the constraints. However, if the arrow moves, neither the box nor the circle moves — the constraints that tie the arrow to the box or circle are violated or removed. If these constraints were multiway, then the constraint solver would move the box or circle when the arrow moved, thus satisfying the constraints.

Multiway constraints are obviously more powerful than one-way constraints, but this increased expressiveness comes at a price. One-way constraint solving algorithms only have to evaluate constraints, making them simpler to implement and more efficient than multiway constraint solvers that must also choose which variable in a constraint should be modified.

Multiway constraints can also introduce ambiguity at the design level. For example, suppose we have the constraint $A - B - C = 0$. If A changes, the constraint solver must choose whether to change B, C, or both. In this case, we should adhere to the principle of least astonishment — a user's editing operation should change the result in a way consistent with the user's expectations.[1,2] One approach to eliminate this ambiguity — constraint hierarchies[2] — divides constraints into hierarchies or priority levels. The constraint solver then tries to satisfy as many constraints as possible, solving the highest priority constraints first, then the next highest, and so on. One issue that constraint hierarchies do not address is how to specify that multiple values should change (for example, when both B and C should change when A changes).

Constraint solving

Constraints can be solved in either a lazy or eager fashion. Lazy evaluation evaluates a constraint only if it affects a result

the Garnet interface builder can display a prototype on the screen and let the user edit it. All instances of that prototype automatically reflect these edits. For example, if a designer changes the standard look and feel of a menu prototype, all menus in the system immediately change accordingly. In most class-instance systems, changing the class structure when instances exist is either very expensive or makes the instances invalid.

A potential disadvantage of the prototype-instance model is that getting a slot's value could require a search up the entire inheritance hierarchy, since slots must be inherited from the prototypes if they are not present in an object. Garnet alleviates this problem by keeping local caches of inherited values. Another efficiency problem is that, since new slots can be added at any time, a fixed storage scheme like a structure or record cannot be used. Rather, a dynamic list of slot names and slot values must be kept. Garnet addresses this problem by keeping the most commonly accessed slots in a Lisp record structure so they can be accessed quickly, and by keeping any new slots in a list.

The efficiency of Garnet's object system is actually better than other Lisp object systems, such as CLOS. For example, on a Sun 3/60 workstation running Allegro Common Lisp, the simplest slot accessing function takes 49.8 microseconds in Portable Common Loops (an implementation of CLOS), but only 27.3 microseconds in KR. Creating an instance in CLOS (16,160 microseconds) takes about 15 times longer than in Garnet (1,117 microseconds).

that the user requests; eager evaluation evaluates a constraint immediately when values change. Thus, a lazy-evaluation system can contain variables whose values are out of date. Lazy evaluation avoids unnecessary work if relatively few values are needed to compute the result the user requests. For example, if portions of a drawing are off screen, they might not have to be recalculated. However, lazy evaluation also introduces extra bookkeeping, since the constraint solver must keep track of out-of-date variables. In addition, lazy evaluation can result in potential delays when the values of out-of-date variables are demanded. Lazy evaluation is most effective in applications where the user wants to view only a limited portion of the application's data and where changes are occurring to all parts of the application's data. Otherwise, eager evaluation is preferable, since it is conceptually cleaner than lazy evaluation (everything is always up to date).

Constraint systems

Examples of graphical systems that use constraint technology abound.[3] Sketchpad[4] pioneered the use of graphical constraints in a drawing editor in the early 1960s. Thinglab[1] took constraints a step further and introduced them into the realm of graphical simulation. More recently, Thinglab has been refined to aid in the generation of user interfaces.[2] Both Sketchpad and Thinglab provide multiway constraints. However, most user interface development systems use one-way constraints because of their simplicity and efficiency; Grow,[5] Peridot,[6] and Apogee[7] are three examples. Grow was the first comprehensive user interface development system to use constraints. Peridot was the first to try to infer constraints. Apogee was the first to employ lazy evaluation. Constraint[8] provided a user interface development environment that used multiway constraints and introduced a new constraint-solving algorithm that made multiway constraints efficient enough to be solved in real time.

The Garnet way

Garnet currently provides one-way constraints and uses lazy evaluation. Whenever a user changes a variable, all constraints that directly or indirectly depend on that variable are marked as out of date. When a user requests the value of an out-of-date variable, the constraint solver demands the values of all variables that this constraint depends on. If these variables are out of date, their constraints will in turn be evaluated, and so on. Eventually, the constraint solver reaches variables whose values are either atomic (that is, not computed by a constraint) or up to date, at which point the constraint solver can work its way back to the originally requested variable.

Future versions of Garnet will probably provide a limited form of multiway constraints and eager evaluation. We have found that almost all constraints in interfaces created with Garnet have to be reevaluated when the display is updated; thus lazy evaluation does not avoid the evaluation of enough constraints to justify the increased bookkeeping it requires. Also, the preliminary design for the new multiway algorithm seems to be very efficient.

References

1. A. Borning, "The Programming Language Aspects of Thinglab: A Constraint-Oriented Simulation Laboratory," *ACM Trans. Programming Languages and Systems*, Vol. 3, No. 4, Oct. 1981, pp. 353-387.
2. B.N. Freeman-Benson, J. Maloney, and A. Borning, "An Incremental Constraint Solver," *Comm. ACM*, Vol. 33, No. 1, Jan. 1990, pp. 54-63.
3. A. Borning and R. Duisberg, "Constraint-Based Tools for Building User Interfaces," *ACM Trans. Graphics*, Vol. 5, No. 4, Oct. 1986, pp. 345-374.
4. I.E. Sutherland, "Sketchpad: A Man-Machine Graphical Communication System," *AFIPS Spring Joint Computer Conf.*, 1963, pp. 329-346.
5. P. Barth, "An Object-Oriented Approach to Graphical Interfaces," *ACM Trans. Graphics*, Vol. 5, No. 2, Apr. 1986, pp. 142-172.
6. B.A. Myers, *Creating User Interfaces by Demonstration*, Academic Press, Boston, 1988.
7. T.R. Henry and S.E. Hudson, "Using Active Data in a UIMS," *Proc. ACM SIGGraph Symp. User Interface Software*, Oct. 1988, pp. 167-178.
8. B. Vander Zanden, "Constraint Grammars — A New Model for Specifying Graphical Applications," *Proc. Conf. Human Factors in Computing Systems (SIGCHI 89)*, Apr. 1989, pp. 325-330.

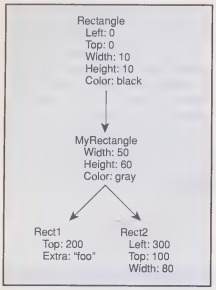
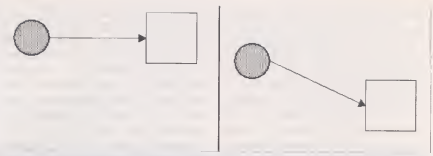


Figure 4. Inheritance in a prototype-instance model. Rect1 and Rect2 are instances of MyRectangle, which is an instance of Rectangle. MyRectangle inherits the left and top of Rectangle. Rect1 inherits all the values of MyRectangle except top, which it overrides. It also adds a new slot. If the left of Rectangle changes, then the lefts of MyRectangle and Rect1 immediately change also. Even though they may be prototypes for other objects, all of these are "real" objects in that they can be displayed on the screen.



```
(create-instance 'myline arrow-line
  (:x1 (formula (gv circle1 :right)))
  (:y1 (formula (gv circle1 :center-y)))
  (:x2 (formula (gv box1 :left)))
  (:y2 (formula (gv box1 :center-y))))
```

Figure 5. The line stays attached to the box and circle even when they move. Below the graphic is the code to define the constraints on the line.

Constraint system. A constraint is a relationship among objects that is maintained when any of the objects changes. Constraints are a natural way to express common relationships in graphical user interfaces. For example, in an editor that supports boxes attached by arrows, the designer could specify a constraint that the arrows must stay attached to the boxes. Then, when the boxes are moved by a program or the mouse, the system automatically moves the arrows as well. (The sidebar discusses constraints and their implementation in more detail.)

Coral[2] was an early version of Garnet's constraints, but it was abandoned because it wasn't fast or flexible enough. Constraints are now integrated with the KR object system.

Constraints in Garnet are arbitrary Common Lisp expressions, stored in object slots. When a program accesses a slot, it cannot tell whether the slot contains a simple value (like a number) or a constraint that calculates the value. Within constraints, references to other objects' slots use the form "(gv other-object slot-name)," where "gv" stands for "get value." Whenever the value of the referenced slot changes, the formula is reevaluated. For example, the code in Figure 5 could be used to keep an arrow attached to two objects.

These formulas are one-way constraints; if the other object changes, the object with the formula is reevaluated, but not vice versa. For example, when the circle or box in Figure 5 moves, the line also moves; however, if the line moves, the circle and box do not move. This type of constraint is also used in Peridot,[8] Apogee,[10] and many other constraint-based user interface systems. As a special case, Garnet can handle cycles in the constraint dependencies (so object *A* can depend on *B*, and *B* can depend on *A*), in which case Garnet goes around the loop exactly once.

Other systems have used more powerful multiway constraints, which we might add to Garnet later. The primary advantage of the current scheme, however, is that constraint evaluation is very efficient. Garnet can reevaluate more than 3,500 constraints per second on the IBM RT PC implementation, which means that many objects with dozens of constraints can be updated in real time as objects are dragged with the mouse. Of course, Garnet ensures that formulas whose values do not change are not reevaluated, so user interfaces can contain many thousands of constraints.

An interesting and novel feature of Garnet's constraints is that the object referenced in the constraint can be accessed indirectly through a variable. For example, a feedback object in a menu might be constrained to be the same size as whatever object it is on top of. A slot would hold the object that the feedback should appear over. Whenever this slot changes, Garnet automatically reevaluates the formulas that depend on the slot, thus moving the feedback object. To make indirection easy, the gv function can be passed a list of slots to use as indirect references. For example, (gv :SELF :other-obj :left) means: "Look in this object's :other-obj slot. The contents of the slot will be another object. Go to that object and get its left." Since this form is common, (gv-indirect . . .) can be used instead of (gv :SELF . . .).

As an example of indirection, the reverse video rectangle in Figure 6 moves whenever the value of the :obj-over slot changes. This mechanism lets designers keep the input-handling specification independent of the graphics. For example, the interactor object that handles menu behavior (described in the "Input handling" section) simply sets the :obj-over slot to the object that the mouse is over, and the constraints ensure that the graphics that handle feedback change appropriately. The interactor does not need to know anything about the appearance of the graphics.

Because any slot of any object can contain constraints, the use of constraints in Garnet is not limited to the position and

size of graphical objects. Constraints are used throughout the system to control many kinds of behaviors. For example, a constraint can be used to set the mode to determine whether an object will change size or grow based on which mouse button is pressed:

```
(:grow-the-object
  (formula (eq (gv-indirect :which-button)
               :rightdown)))
  ; grow if right button is pressed,
  ; otherwise move
```

Constraints can also be used to connect graphical objects to application-specific objects. For example, the value of a gauge displayed on the screen could be constrained to a temperature data value in the application.

Graphical object system. Opal, Garnet's graphical object system, is designed to make creating and editing graphical objects easy. Opal provides default values for all object properties, so simple objects can be drawn by specifying only the necessary parameters. For example, to create a rectangle at (10,20) with size 30 by 40, a designer need only write:

```
(create-instance 'myrect rectangle
  (:left 10) (:top 20) (:width 30)
  (:height 40))
```

The object system is integrated with the constraint system, so that any object property can be specified with formulas instead of numbers, as shown in the code in Figure 5.

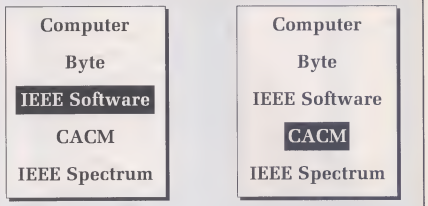
Opal also automatically handles object drawing and erasing. If any object property changes, Opal automatically refreshes the screen and redraws that object. If that object overlaps others on the screen, Opal ensures they are also redrawn correctly (see Figure 7). In addition, if the modifications change other objects due to constraints, these other objects are also redrawn automatically.

Rather than simply redrawing all the objects, Opal tries to minimize the number of objects that are erased and redrawn. This is very important for complex scenes. For example, moving one object through a window containing 200 other objects takes 25.6 milliseconds per move (39 moves per second) rather than the 568 milliseconds (1.76 moves per second) it would take if Opal redrew all the objects.

Because Opal knows where all objects are, it can also handle window refresh (when the window is uncovered). Opal's clients never have to worry about the window system's refresh events. In fact, Opal and the interactors completely hide all window system functions.

To make an object appear in Opal, the designer only has to add it to an Opal window. Removing the object from the window makes the object disappear. To change an object's color or size, the designer sets the appropriate slots. Therefore, the programmer never calls the draw or erase methods on objects directly; these methods are only called from internal Opal routines. In this respect, Opal departs significantly from other graphical object systems.

An important aspect of Opal is its ability to group graphical objects into collections called *aggregates*. When an aggregate is used as a prototype, its instances contain copies of the entire collection of objects. Thus, there is an instance for each component of the aggregate as well as for the aggregate itself (see Figure 8). Changes to the aggregate are immediately reflected in




```
(create-instance 'feedback-box Rectangle
  (:obj-over NIL)                          ; The object that should be
                                           ; highlighted
  (:visible (formula (gv-indirect :obj-over)))  ; I am visible if there is an object
                                           ; in :obj-over
  (:left (formula (gv-indirect :obj-over :left)))  ; The size and position is the same
                                           ; as whatever I am over
  (:top (formula (gv-indirect :obj-over :top)))
  (:width (formula (gv-indirect :obj-over :width)))
  (:height (formula (gv-indirect :obj-over :height)))
  (:draw-function :xor))                   ; XOR this rectangle
```

Figure 6. Setting the :obj-over slot of the feedback-box rectangle makes it appear over that object because of the constraints on the size and position.

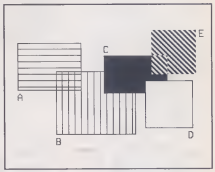


Figure 7. Opal lets graphics overlap opaquely and automatically refreshes damaged parts. If C is erased, B, D, and E will need to be redrawn. Opal uses a clipping region so that only the parts of B, E, and D that overlap C are affected, and so that A does not have to be redrawn. B is drawn with the OR drawing function, E is drawn with XOR, and the others are drawn with Copy.

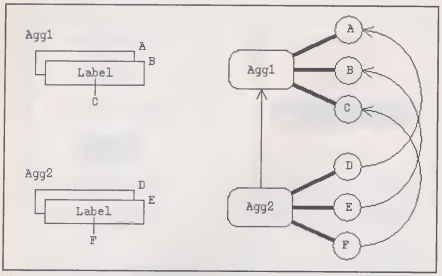
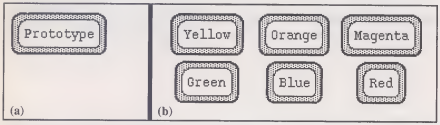


Figure 8. A, B, and C are components of the aggregate Agg1. Making an instance of Agg1 automatically makes instances of each of its components. The dark lines are component links, and the arrow lines are instance links. If Agg1 or any of its components change, Agg2 and its components would change automatically.




```
(create-instance 'colorlist AggreList
   (:left 150)(:top 10)
   (:Item-Prototype My-Roundtangle-Prototype)
   (:Items '("Yellow" "Orange" "Magenta" "Green" "Blue" "Red"))
   (:direction :horizontal)
   (:h-align :center)                    ; Center the objects in the field
   (:rank-margin 3)                      ; Go to next line after 3rd object
   (:fixed-width-p T))                   ; All fields have same width
```

(c)

Figure 9. A prototype for the menu items (a) and a two-dimensional aggrelist using instances of that prototype for each element (b). The actual code to display the menu is shown in (c). If the prototype is subsequently edited to add a drop shadow and change the font (d), all instances immediately inherit the new values and structure (e).

all instances, including when components are added or deleted from the prototype, in which case the corresponding components are immediately added or deleted from all instances. Previous implementations of the prototype-instance model have not supported such structural changes to instances.

As an example, consider a graphical tool that lets the user design menus. The user might draw a prototype button (Figure 9a) and then create many instances of that button in the interface (Figure 9b). If the prototype is subsequently edited — perhaps to add a drop shadow and change the text font (Figure 9d) — Opal insures that all instances immediately inherit the changes (Figure 9e). This requires that Opal add a new object for the shadow to each instance.

Of course, each instance can override any of the slots, making it easier to build prototypes for complex, composite objects like menus and scroll bars. The prototype contains example values for slots (like the text strings in the menu), and the instances override these values.

A special form of aggregate is an *aggrelist*, which is used to lay out a list or table of elements. An aggrelist contains a single prototype for all the elements, and each element uses a different value for some value of that prototype. For example, menus are usually implemented as an aggrelist with a single prototype for the items but with a different string value displayed in each item. The prototype in Figure 9a is an aggregate containing the two rounded rectangles (one grey and one white) and the example string "Prototype." Aggrelists allow the items to be displayed horizontally, vertically, and in multiple rows, or the programmer can specify a layout (as for the gauge labels shown later in Figure 12f). We are also working on the implementation of aggregraphs and aggretrees to lay out graphs and trees.

Input handling. One of the most difficult tasks when creating highly interactive user interfaces is handling the mouse, keyboard, and other input devices. Window managers and user interface toolkits typically provide only a stream of device-dependent mouse positions and keyboard events, and they require that the programmers handle all interactions themselves. Garnet provides significantly more help through the use of interactors, which are encapsulations of input device behaviors.[3]

There are few distinct behaviors in user interfaces. For example, although graphics can vary significantly and the specific mouse buttons can change, all menus operate in

essentially the same manner. Another example is the way objects move when being dragged with the mouse. The interactors capture these common behaviors in a central place while still allowing a high degree of customizing by application programs.

Other advantages of the interactors are:

- They are entirely look independent; any graphics can be attached to a particular feel (see Figures 10 and 11).
- They let designers separate the details of object behavior from the application and the graphics (long a goal of user interface software design).
- They support multiple input devices operating in parallel.
- They let users operate on any of a number of different applications running in separate windows in the same Lisp process and same address space, thereby supporting a form of multiprocessing.
- Opal and the interactors hide the complexities of X Windows graphics and event handling, making Garnet easier to use than X Windows and allowing Garnet to be ported to other graphics packages, such as Macintosh Quick Draw or Display Postscript, without changing Garnet applications.

There are only six types of interactors in Garnet, and these cover all the kinds of interactions used in graphical user interfaces:

- *Menu-Interactor*. For choosing one or more items from a set, or for a single stand-alone button.
- *Move-Grow-Interactor*. For moving or changing the size of an object or one of a set of objects using the mouse. This interactor can be used for one- or two-dimensional scroll bars and horizontal and vertical gauges, and for moving or growing application objects in a graphics editor.
- *New-Point-Interactor*. For entering one, two, or an arbitrary number of new points using the mouse; for example, for creating new lines or rectangles in an editor.
- *Angle-Interactor*. For calculating the angle at which the mouse moves around some point. It can be used for circular gauges or for rotating objects.
- *Trace-Interactor*. For capturing all the points the mouse goes through between start and end events, as needed for free-hand drawing.
- *Text-String-Interactor*. For inputting a small (optionally multiline) string of text.

Each interactor is parameterized in various ways, so the programmer can control the mouse or keyboard events that make it start and stop and the optional application procedures to be called on completion. The most significant parameters, however, are the objects where the interactor operates and the (optional) objects to handle feedback. Each type of interactor controls the graphics with a well-defined protocol. For example, the menu interactor sets the :obj-over slot of the feedback object. Therefore, to turn the list of items in Figure 9b into a menu using the feedback object from Figure 6, we need only the following code:

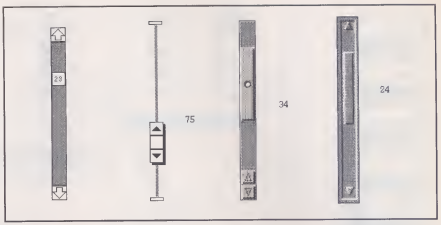


Figure 10. The same type of interactor can handle different graphic looks. Here, move-grow-interactors for the indicators and menu-interactors for the arrow buttons handle scroll bars that look like those in the Macintosh, Open Look, Next, and OSF/Motif environments.

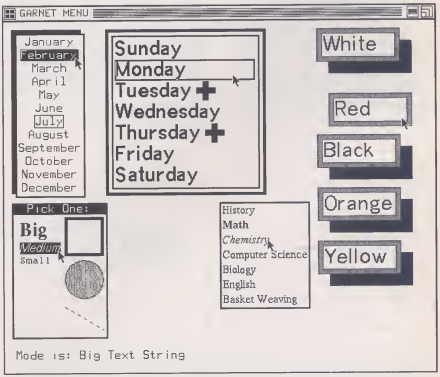


Figure 11. The menu-interactor can handle menus with many different looks and feels. The items in (c) simulate floating by moving when the mouse is over them. The items in (e) change to italic when the mouse is over them; the final selection is bold.

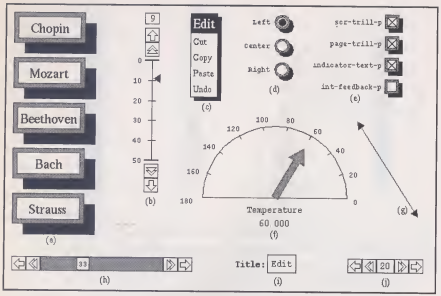


Figure 12. Some of Garnet's gadgets: (a) floating buttons, (b) a number slider, (c) a menu, (d) floating radio buttons, (e) floating check boxes, (f) a semicircular gauge, (g) an arrow-line, (h) a scroll bar, (i) a labeled text entry field, and (j) a number entry field.

```
(create-instance 'color-selector
        menu-interactor
  (:start-where '(:element-of colorlist))
    ; Colorlist is defined in Figure 9c
  (:feedback-obj feedback-box))
    ; Feedback-box is defined in Figure
    ; 6 and will appear over whichever
    ; of the items the mouse is over
```

The feedback can be a set of objects, for example, the selection handles in Figure 1. As shown in Figure 11, the feedback does not even have to be a separate object. Instead, some property of the object itself can change in response to the mouse. If the :feedback-obj is not supplied, the menu interactor sets the :interim-selected slot of the object the mouse is over and the :selected slot of the item the mouse ends up on. The menu in Figure 11c has constraints that change the position of the object under the mouse, based on whether the value of :interim-selected is T or NIL (true or false). The menu in Figure 11e decides whether to use a Roman, bold, italic, or bold-italic font based on the values of both :interim-selected and :selected.

An important feature of interactors is that a single interactor can handle a set of objects. For example, there is a single interactor for each menu, rather than one for each menu item. Similarly, a single move-grow interactor can be used for an entire set of objects that can be moved with the mouse (as in Figure 1). The interactor will choose which object to move based on where the mouse button is pressed.

The interactors are based on Smalltalk's Model-View-Controller.[12] In Garnet, the application objects correspond to the model, the Opal graphical objects correspond to the view, and the interactors correspond to the controller. In Smalltalk, however, if a programmer wants a new style of object, he or she usually has to write code for all three parts. In Garnet, programmers almost never need to create new forms of interactors, even for application-specific objects; it is sufficient to create an instance of one of the supplied types of interactors and then supply appropriate parameters to achieve the desired behavior. New types of interactors would be needed in Garnet only for radically different types of behaviors, such as gesture recognition or new kinds of physical input devices (such as a 3D joystick). Constraints in Garnet connect the models, views, and controllers.

Gadgets. On top of Opal and the interactors is a collection of interaction techniques — called gadgets — that provide a starting point for applications. There are gadgets for menus, scroll bars, buttons, gauges, etc. Figure 12 shows some of the gadgets supplied with Garnet (more are being created). These can be used by designers who do not want their applications to have a custom look and feel. Most of the gadgets have a number of parameters to let designers vary many aspects of the appearance and behavior.

In some toolkits, such as Xtk and Interviews, each gadget is a window. This significantly limits what gadgets can do. In Garnet, however, gadgets are not windows, so there is a gadget, for example, that is a line with an attached arrowhead (Figure 12g). An even more sophisticated gadget in Garnet handles object selection (see Figure 13). It displays selection handles around the selected object and lets the user move or grow the object by pressing on the handles. Therefore, to support graphical object selection, a designer using Garnet need only create an instance of an object-selec-

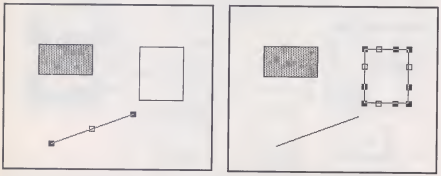


Figure 13. The graphics selection gadget shows control handles around the selected object. Pressing on a white handle moves the object, and pressing on a black one changes the object's size. When a line is selected, only three control points are shown: the black ones change the end points and the white one moves the line, keeping the same length and slope.

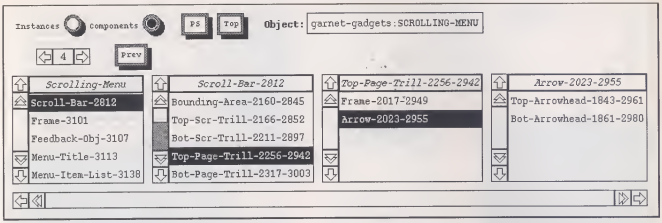


Figure 14. A browser that will show either the components of an aggregate or the instances of a prototype. We created this with the Garnet Toolkit, and it can be used to debug Garnet code.

tion gadget (assuming he or she finds the Garnet look and feel acceptable).

Since creating new gadgets with Garnet is easy, we expect many designers to create their own gadgets rather than use the supplied ones. By using Lapidary (detailed in the next section) or by programming using Opal and interactors, programmers can create new styles of menus and buttons in minutes.

Debugging tools. Defining user interfaces with constraints and interactors is natural and effective, but it can be difficult to find bugs in the code. Since with constraints, a bad value in one place can cause many different objects to have bad values, the effects of a bug are not local.

Garnet provides a large and growing number of debugging tools to help with program implementation. These tools provide tracing mechanisms to show how a slot got its current value, where objects are on the screen or why they are not visible, and what happens when the mouse or keyboard is used. The tools also provide extensive checking for legal values of graphics slots and convenient ways to inspect objects, including browsers (see Figure 14).

The interface builder

Lapidary. On top of the Garnet Toolkit layer are a number of tools to make creating user interfaces easier. The most important is the Lapidary interface builder.[4] Lapidary provides a graphical front end to most of the underlying toolkit features, so that a program's graphical aspects can be specified pictorially. In addition, the behavior of these objects at runtime can be specified using dialogue boxes and by demonstration.

In particular, Lapidary lets the designer, who does not have to be a programmer, draw pictures of application-specific graphical objects that the application will create and maintain at runtime. These objects include the graphical entities the end user will manipulate (such as the components of the picture), the feedback showing which objects are selected (such as small squares that serve as handles on the sides and corners of an object), and the dynamic feedback objects (such as hairline boxes to show where an object is being dragged). The designer creates prototypes of the objects in Lapidary, and the application program then creates instances of the prototypes as needed.

In addition, Lapidary supports the construction and use of gadgets, such as menus, scroll bars, buttons and icons. Lapidary therefore supports both the use of a predefined library of gadgets and the definition of new gadgets with a unique look and feel. Both types can even be combined in the same application. The designer can specify the runtime behavior of these objects in a straightforward way using constraints and abstract descriptions of their interactive response to the input devices. Lapidary generalizes from the specific example pictures to let the designer specify the graphics and behaviors by demonstration.

The designer can specify the behavior of objects in various ways. Graphical constraints can be attached to objects using iconic menus (see Figure 15). If an object should move with the mouse, it can be selected and declared a feedback object. Lapidary will automatically generalize the constraints on the feedback object so that they refer to whatever graphical object the mouse is over. For example, in Figure 15b the check mark is constrained to the Stop button, but Lapidary generalizes this constraint to use a variable (like the :obj-over slot of Figure 6) so that the check mark can appear next to any of the buttons.

Also, if an object should change based on some user action, the designer can specify this by demonstration. The designer draws one state, then another, and Lapidary automatically constructs the constraints to change the object between the two states. The menus of Figures 11c and 11e can be defined in this way.

Consider how a designer might create the boxes and arrows for a graph editor. First, the designer draws a picture of the boxes, say, using rounded rectangles (Figure 16a). Then, he or she creates an example text label and uses the iconic constraint menus to center it at the top (Figure 16b). This will serve as the prototype for the boxes the application will create, so the designer selects the entire group and saves it as a named prototype. Lapidary asks the designer for the object parameters, so the designer specifies the position and size of the box and the string for the label. Lapidary then writes this prototype to a file and defines it in memory (in case the application is running concurrently with Lapidary).

Next, the designer creates a copy of the

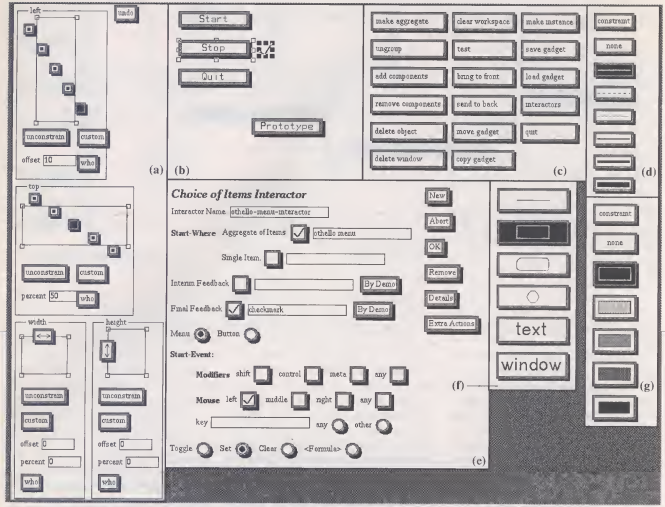


Figure 15. Lapidary in action. The work window (b) contains a prototype and three buttons made from it. The check mark icon is the primary selection (indicated by black squares) and the center button is the secondary selection (indicated by white squares). The iconic constraint menu (a) shows that the primary selection is constrained to be 10 pixels to the right of the secondary selection and centered by it vertically. Window (c) contains the main Lapidary commands. The menus on the right determine the type of the next object to be created (f), the line style (d), and the fill style (g). The large window in the center (e) is a dialogue box for specifying the behavior of the menu being created; it shows that the check mark icon will be the final feedback to show which item is selected when the left mouse button is pressed. This dialogue box was created using Jade.

box, draws an arrow-line (Figure 16c), and uses the line constraint menu to specify that the ends of the line should be centered in the boxes (Figure 16d). Then the designer selects only the line and saves it as a prototype. Lapidary notes that the prototype has constraints to objects that are not being saved, so it asks the designer if the objects are to be parameters of the line prototype. The designer specifies that the parameters should be called from-obj and to-obj. Now the application can create instances of the lines, supplying the objects for the from-obj and to-obj slots, and new lines will appear in the application window. The application knows nothing about the graphics used for the lines, and the designer did not have to write any graphics code. This entire design session takes about four minutes.

Jade. It is sometimes easier to list the contents of a dialogue box or menu than to meticulously draw it. The Jade dialogue box creation system automatically creates a dialogue box or menu from a specification of its contents.[5] In addition to being simple to work with, the specification passed to Jade has the additional advantage of being independent of any particular look and feel. The textual specification has a Lisp-like syntax and lists only the string labels to appear in the dialogue box, the type of input required (such as the choice of one item of a set, or of a number in a range). When Jade is given a particular look and feel (Macintosh, Motif, Garnet standard, etc.), it can choose the correct interaction techniques, which themselves can be designed using Lapidary. The heuristic rules that determine the placement of various parts of the interface are specific to a particular look and feel. For example, the set of buttons that make a dialogue box go away ("OK," "Cancel," etc.) will be at the

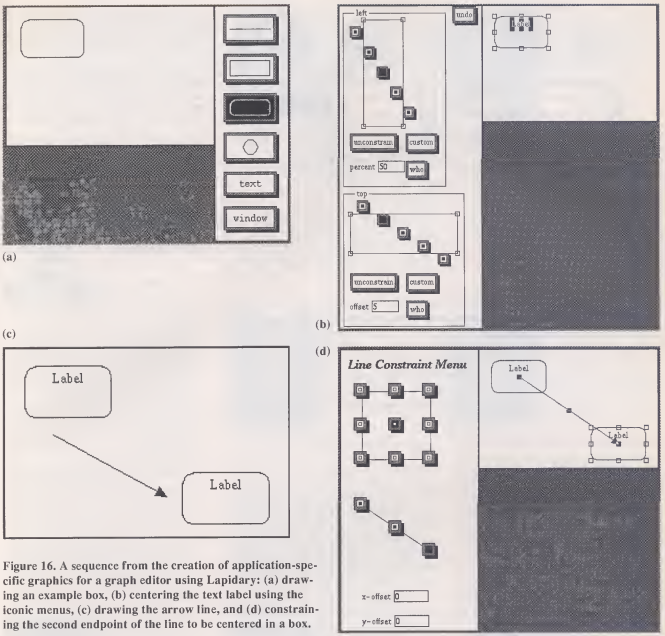


Figure 16. A sequence from the creation of application-specific graphics for a graph editor using Lapidary: (a) drawing an example box, (b) centering the text label using the iconic menus, (c) drawing the arrow line, and (d) constraining the second endpoint of the line to be centered in a box.

right for a Macintosh-like dialogue box and at the top for a Xerox Star-like one. Jade created the dialogue box in Figure 15e automatically.

The heuristics for placing objects in dialogue boxes attempt to create an attractive display based on graphic arts principles. If the designer is not happy with the dialogue box, however, he or she can edit it in Lapidary, adding decorations (such as extra rectangles and lines) and moving parts around. These changes are saved in an exceptions file so they can be reapplied even if the original textual specification is edited and the dialogue box regenerated.

C32 spreadsheet. Although many desired constraints can be specified using Lapidary's iconic menus, designers occasionally need more powerful constraints. The C32 spreadsheet program in Garnet lets designers enter arbitrary Lisp constraint expressions. It provides many of the advantages for graphics that financial spreadsheets provide for business. In particular, C32 lets the designer monitor and debug interfaces by watching spreadsheet values while the user interface is running. It also provides extensible command menus, automatic generation of appropriate object references from mouse clicks in graphics windows, and automatic generalizations of example constraints so they can be used in multiple formulas. Figure 17 shows a typical C32 session.

The Garnet Toolkit is operational and has many local and external users. More than 80 companies and universities are licensed to use Garnet. (Garnet is available free from Carnegie Mellon University. Designers interested in using Garnet should contact Myers at the CMU address listed after the biographies.) We are working to increase the toolkit's

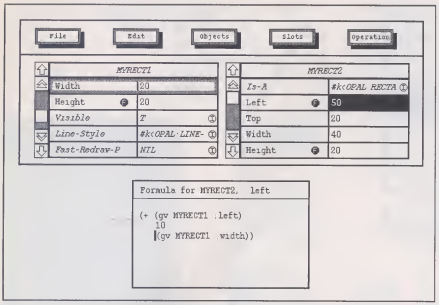


Figure 17. Viewing two objects in the Garnet C32 spreadsheet. At the top of the window is a set of pull-down menus. Each object has its own set of columns. The slot names of the objects are at the left of each column, along with the current values. If the value is computed by a formula, an icon with an "F" is shown. Inherited values are in italics and marked with a different icon. At the bottom is a formula display window for the selected slot. When the user clicks on a slot or an object in a graphics window, a reference to that object is inserted into the formula.

functionality and performance and to find new ways to support the design and implementation of application user interfaces. We hope the toolkit will be widely distributed and used in the Common Lisp community. Lapidary is working but is not yet releasable. Jade is partially working and has been used to generate the Lapidary dialogue boxes. The C32 system is just now being implemented.

We plan to work on allowing more of the interface to be specified by demonstration rather than through dialogue boxes and coding. For example, Lapidary could infer graphical constraints and mouse dependencies from the user's drawing, in a manner similar to Peridot.[8] We also hope to extend Garnet to handle other input and output technologies, such as physical dials and switches, speech input and output, and gesture recognition.

Finally, we plan to provide a high-level graphical editor framework that provides many of the operations common in systems that let the user create and manipulate graphical objects. These might include support for a palette of object types; selecting objects; changing object size, position, and other properties; deleting objects; automatically laying out objects as a list, table, graph, or tree; saving and restoring objects to a file; printing; and undoing. For many programs, this subsystem will provide most of the standard functionality, and designers will need to specify only the application-specific parts.

Garnet has only been working for a short time, but it has already demonstrated that it makes the creation of graphical, highly interactive user interfaces easier. For example, in an informal experiment, creating a simple graphical editor like that in Figure 1 took various people two to four hours with the Garnet Toolkit but 10-20 hours with other toolkits such as Apple's MacApp and Sun's Open Look. When Lapidary and the other high-level tools are released, we expect programmer productivity to increase even more. ■

Acknowledgments

In addition to the authors, others who have helped design and implement Garnet include Pedro Szekely, Pavan Reddy, Jake Kolojejchick, and Lynn Baumeister. Amy Moormann Zaremski and Pedro Szekely were brave early users of the Garnet Toolkit and helped us debug it. Bernita Myers, Amy Moormann Zaremski and the referees provided helpful comments on this article.

This research was sponsored in part by the Defense Advanced Research Projects Agency, under contract F33615-87-C-1499, ARPA Order No. 4976, Amendment 20, monitored by the Avionics Laboratory, Air Force Wright Aeronautical Laboratories, Aeronautical Systems Division (AFSC), Wright-Patterson AFB, Ohio 45433-6543.

References

1. B.A. Myers, "User Interface Tools: Introduction and Survey," *IEEE Software*, Vol. 6, No. 1, Jan. 1989, pp. 15-23.

2. P.A. Szekely and B.A. Myers, "A User Interface Toolkit Based on Graphical Objects and Constraints," *Proc. ACM Conf. Object-Oriented Programming, Systems Languages, and Applications in SIGPlan Notices*, Vol. 23, No. 11, Nov. 1988, pp. 36-45.

3. B.A. Myers, "Encapsulating Interactive Behaviors," *Proc. Conf. Human Factors in Computing Systems (SIGCHI 89)*, Apr. 1989, pp. 319-324.

4. B.A. Myers, B. Vander Zanden, and R.B. Dannenberg, "Creating Graphical Objects by Demonstration," *Proc. ACM SIGGraph Symp. User Interface Software and Technology*, Nov. 1989, pp. 95-104.

5. B. Vander Zanden and B.A. Myers, "Automatic, Look-and-Feel-Independent Dialogue Creation for Graphical User Interfaces," *Proc. Conf. Human Factors in Computing Systems (SIGCHI 90)*, Apr. 1990, pp. 27-34.

6. B.A. Myers et al., "The Garnet Toolkit Reference Manuals: Support for Highly Interactive, Graphical User Interfaces in Lisp," Tech. Report CMU-CS-90-117, Carnegie Mellon University, Computer Science Department, Mar. 1990.

7. M.A. Linton, J.M. Vlissides, and P.R. Calder, "Composing User Interfaces with Interviews," *Computer*, Vol. 22, No. 2, Feb. 1989, pp. 8-22.

8. B.A. Myers, *Creating User Interfaces by Demonstration*, Academic Press, Boston, 1988.

9. A. Borning, "Thinglab — A Constraint-Oriented Simulation Laboratory," Tech. Report SSL-79-3, Xerox Palo Alto Research Center, July 1979.

10. T.R. Henry and S.E. Hudson, "Using Active Data in a UIMS," *Proc. ACM SIGGraph Symp. User Interface Software*, Oct. 1988, pp. 167-178.

11. C. Wiecha et al., "Generating User Interfaces to Highly Interactive Applications," *Proc. Conf. Human Factors in Computing Systems (SIGCHI 89)*, Apr. 1989, pp. 277-282.

12. G.E. Krasner and S.T. Pope, "A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System," *J. Object-Oriented Programming*, Vol. 1, No. 3, Aug. 1988, pp. 26-49.

Brad A. Myers is a research computer scientist at Carnegie Mellon University, where he is the principal investigator for Garnet. His research interests include user interface development systems, user interfaces, programming by example, visual programming, interaction techniques, window management, programming environments, debugging, and graphics.

Myers received the MS and BSc degrees from the Massachusetts Institute of Technology, and he received a PhD in computer science at the University of Toronto, where he developed the Peridot user interface management system. He is a member of the IEEE, the Computer Society, the ACM, SIGGraph, and SIGCHI.

Brad Myers can be reached at the School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213-3890. Electronic mail can be sent to garnet@cs.cmu.edu.

Roger B. Dannenberg is a research computer scientist at Carnegie Mellon University. His research interests include programming language design and implementation and the application of computer science techniques to the generation, control, and composition of computer music.

Dannenberg received a BA from Rice University in 1977, an MS from Case Western Reserve University in 1979, and a PhD from Carnegie Mellon in 1982. He is a member of Phi Beta Kappa, Sigma Xi, Tau Beta Pi, Phi Mu Alpha, the ACM, SIGCHI, and the Computer Music Association.

Brad Vander Zanden is an assistant professor at the University of Tennessee and an active participant in the Garnet project. He was a postdoctoral fellow at Carnegie Mellon University for two years. His interests include user interface development systems, user interfaces, constraint systems, incremental constraint satisfaction, programming environments, and graphics.

Vander Zanden received his BS degree from Ohio State University, and his MS and PhD degrees from Cornell University. He is a member of the IEEE, the Computer Society, the ACM, SIGGraph, and SIGPlan.

Dario A. Giuse is a senior computer scientist at Carnegie Mellon University. His research interests include user interface development environments, constraint satisfaction, graphics, knowledge representation systems, tutoring systems for foreign languages, medical informatics, and interactive knowledge acquisition systems.

Giuse received the Dr.Ing. degree (summa cum laude) in computer science from the Politecnico di Milano, Italy, in 1979. He is a member of the IEEE Computer Society, the ACM, the AAAI, and the Mathematical Association of America.

David S. Kosbie is a graduate student at Carnegie Mellon University, where he is a member of the Garnet project. His research interests include user interface development systems, user interfaces, and methods for simplifying graphical programming. Kosbie received the BA degree from Harvard University in 1987. He is a member of the IEEE and the Computer Society.

Edward Pervin is a research programmer at Carnegie Mellon University, where he is working on Garnet. Pervin received his MS and BSc degrees in mathematics from Carnegie Mellon.

Andrew Mickish received a BS degree from Carnegie Mellon University, where he worked on the Garnet project as an undergraduate. He plans to enter graduate school to pursue interests in user interfaces and artificial intelligence.

Philippe Marchal is a member of the Human-Computer Interaction group of the European Computer-Industry Research Center in Munich. He is a former visiting scientist at Carnegie Mellon University, where he was a researcher in the Garnet project. His research interests include user interfaces, user interface management systems, interaction techniques, human factors, graphics, and object-oriented programming. Marchal graduated from the University of Grenoble, France.

CALL FOR PAPERS

Data Compression Conference

(Sponsored by the IEEE Computer Society TCC in Cooperation with NASA/CESDIS)

Snowbird, Utah
April 8 - 10, 1991

Program Committee:
A. Blumer, *Tufts U.*
R. Capocelli, *U. Rome*
J. Cleary, *U. Calgary*
P. Elias, *MIT*
I. Daubechies, *Bell Labs*
R. Gray, *Stanford U.*
D. Hirschberg, *UC Irvine*
A. Lempel, *Technion*
V. Miller, *IBM*
J. Reif, *Duke U.*
D. Sheinwald, *IBM*
J. Storer, *Brandeis U.*
J. Tilton, *NASA*
J. Vitter, *Brown U.*
A. Wyner, *Bell Labs*
J. Ziv, *Technion*

Co-General Chairs:
J. Reif, *Duke U.*
J. Storer, *Brandeis U.*

Publicity Chair:
R. Miller, *NASA/CESDIS*

Registration Chair:
M. Cohn, *Brandeis U.*

THEME: Data compression has become an essential component of high speed communications and mass storage. This conference will provide an international forum for current work on data compression and related areas. Topics of interest include but are not limited to: *Coding theory, quantization theory, parallel algorithms and compression hardware, algorithms for specific types of data (including text, images, video, speech, maps, instrument data, graphics, animation, and bit-maps), transform methods, wavelet and fractal techniques, string searching and manipulation, closest-match retrieval, theory of minimal length encoding and applications to learning, system issues (including error control, data security, and indexing), medical imagery, scientific and space data archives, data compression standards.*

SUBMISSION: Authors are requested to submit 3 copies of an extended abstract of at most 10 pages by **December 14, 1990** to:

Prof. Martin Cohn
Computer Science Dept.
Brandeis University
Waltham, MA 02254
(fax: 617-736-2741, email: marty@cs.brandeis.edu)

NOTIFICATION: Authors will be notified of acceptance or rejection by a letter mailed on 1/15/91. A final draft is due 2/15/91. Students presenting papers are eligible for financial support to attend the conference.

ADVANCE REGISTRATION: Registration includes reception, banquet, coffee breaks, and proceedings. For registrations *received before 3/1/91*, the fee is $295.00 for IEEE or affiliate members or $370.00 for non-members; there is an additional $100 late fee for registrations after 3/1/91.

HOTEL: Space is limited; rooms may **NOT** be available after 3/1/91. Room rates are $42 per person for a double room and $84 for a single room. Call the *Cliff Lodge, Snowbird, UT 84092*, 800-453-3000 or 801-742-2222.

TRANSPORTATION: The hotel may be reached by limousine or public bus from the Salt Lake City airport.

REGISTRATION FORM

NAME: _____

AFFILIATION: _____

ADDRESS: _____

_____

_____

PHONE: _____ EMAIL: _____

IEEE member ☐ yes ☐ no AMOUNT ENCLOSED: _____

MAIL TO:

DCC '91
Att. Myrna Fox
Computer Science Dept.
Brandeis University
Waltham, MA 02254

Editor: Fletcher J. Buckley, 103 Wexford Dr., Cherry Hill, NJ 08003, phone (609) 866-6350, fax (609) 866-7753, Compmail f.buckley

An American's view of the Japanese standards system*

John P. Stern, American Electronics Association

High atop Kasumigaseki, the section of Tokyo that is the heart of Japan's industrial bureaucracy, 20 Japanese electronics experts meet at the Ministry of International Trade and Industry (MITI) to discuss the developing field of optoelectronics. The entire conversation is in Japanese. By the time the deliberations end, Japanese business and government have reached a better understanding regarding the direction of new technologies.

This could be the opening scene in any number of paperback novels about business-government cooperation in Japan, except that for the past few years, two foreigners — one representing the European information industry and the other representing its American counterpart — have also been present in the room and have participated as full, legal committee members.

That the Japanese information technology standards-setting process is open to foreign participation is now fully established, and more than 20 foreign-capital companies in Japan participate. The challenge to foreign information technology companies is effectively using this access in their competitive strategy.

The impact of standardization. Industry standardization in Japan can suddenly unlock a potential market, leaving unprepared companies ill-equipped in the path of massive Japanese competition. Some may remember what happened in the American facsimile machine market 10 years ago. American-owned facsimile machine makers did not agree on a communications standard. Consumers were deterred by an expensive machine that could not communicate with anyone except the user of the identical make.

In 1981, however, Japanese companies adopted the G III standard for facsimile machines, making it possible for one G III machine to communicate with many different makes and models of other G III machines. With some of the uncertainty over consumer acceptance of the facsimile reduced, Japanese fax makers began to mass produce and lower unit costs, leading to lower prices.

Today, most offices in America use their fax machines frequently, and the convenience of being able to send a fax without worrying about incompatibility is widely appreciated. However, there haven't been any American-owned manufacturers of commercial facsimile machines left in the United States for many years.

Standardization in Japan is an immense topic, both in breadth and in detail. The Japan Industrial Standards system, administered by the Ministry of International Trade and Industry, contains more than 8,200 standards covering everything from optical disks to heated toilet seats.

The players. Standardization in Japan is an immense topic, both in breadth and in detail. The Japan Industrial Standards (JIS) system, administered by MITI, contains more than 8,200 standards covering everything from optical disks to heated toilet seats. Overseas, the JIS system is perhaps the best known Japanese standards system, but other aspects of standardization affecting the electronics industry are often under the jurisdiction of the Ministry of Posts and Telecommunications or the Ministry of Health and Welfare.

It is not uncommon to find . :ral Japanese ministries contesting the right to set standards. For example, when a telecommunications trunk line caught fire in Tokyo several years ago, crippling the nationwide fund transfers of a major Japanese bank, several ministries vied for the right to set data network safety, reliability, and security standards. MITI claimed that computer data transfer was in its area of jurisdiction. The Ministry of Posts and Telecommunications claimed that telecommunications was its area of jurisdiction. The Ministry of Finance was interested because electronic fund transfers and the banking system are in its portfolio. The National Police Agency was concerned with the public order aspects of trunk-line security. Even the Ministry of Construction demanded a say, based on its authority over buried cable.

Non-government standards. There are areas of standardization that are not the subject of formal government deliberation procedures because the technology is changing too rapidly to make it worthwhile to try to freeze the standard. An example is surface mount technology. One must follow standards in this area by attending discussion groups that often do not leave an extensive public paper record.

Japan also has its share of domestic voluntary technical standards, which a foreign company is not legally required to adopt, but which can become an issue in customer acceptance. Examples include the electromagnetic emissions standards of the Voluntary Control Council for Interference by Data Processing Equipment and Electronic Office Machines (VCCI) and the Communications Industry Association of Japan's "C" mark for telephone voice quality. (Roughly in the shape of the letter, the "C" mark is a seal of approval that indicates that the telephone equipment con-

* This is an extract of a presentation made to the US National Academy of Sciences in October 1989.

forms to the CIAJ's voice-quality standards.)

Japanese companies do not all agree on standards or technologies. Some companies plan to use a unique standard and lock in customers to their standard with rapid commercialization and superior marketing. Other companies plan to wait until a common standard emerges, then use their mass production and captive sales channel advantages to push ahead of the competition. For example, I have heard standards drafting members from one company flatly tell the group that their company would not be adopting the group's suggested standard size for optical disks.

Current trends. In 1984, foreign participation on Japanese standards bodies was still rare, and there were times when we had some of the experiences from which stories about nontariff barriers are made. In 1985, after extensive discussions with the US concerning liberalization of the Japanese telecommunications market as well as its own research and deliberation, Japan's Ministry of Posts and Telecommunications made a fundamental change in its regulatory policy. That change was to move from an era of government-imposed telecommunications standards to an era of private-sector proposed telecommunications standards.

The flagship of this effort has been the Telecommunication Technology Committee, which since its founding in 1985 has had representatives of foreign-capital telecommunications companies on its board of directors as well as in its working groups. The TTC has evolved from a group that reviews American and International Telegraph and Telephone Consultative Committee network standards for Japan to one that develops pioneering policies in areas like intellectual property protection for network protocols. The US and Europe are still grappling with that issue.

Also in 1985, MITI, as part of a major new standardization plan, made an institutional commitment to internationalize Japan's technical standards procedures. Increasing trade friction provided some of the incentive to open up Japan's technical standardization process, and a pledge to provide transparency and access to foreign companies was included in the July 1985 "Action Program" of the Nakasone Cabinet.

However, even before the Action Program, MITI had determined that complaints about closed Japanese standards-setting procedures were a profoundly unnecessary source of friction, that Japan could well afford to allow more foreign access to its standards-setting process, and that increased international technical exchange would enhance Japan's technological base. MITI also encouraged Japanese companies to use the international standards-setting process, as represented by the International Organization for Standardization and other factions, so as to have the world agree, through representative voting, to make Japanese standards the world standards.

The JIS standards process was still closed to foreigners in 1985 in the sense that Japanese citizenship was required to participate beyond the drafting stage. However, two years later, MITI took the bold step of reforming Japanese government policy to allow foreigners to sit as full members at the technical committee level of JIS. The two foreigners representing American and European electronics interests have been sitting on the Optoelectronics Technical Committee since mid-1987.

The next higher level of JIS, the Electronics Divisional Council level, reviews all electronics standards, not only those in optoelectronics. Membership in this level is open only to Japanese citizens, but I have personally appeared as a witness when the standards being considered were of interest to American industry. The top level of JIS is open only to Japanese citizens but, since it reviews all JIS standards, it is more policy than technical oriented. Of course, Japanese citizens employed by foreign companies are eligible for membership.

While contacts made in the standards-setting process can be useful in marketing, sitting on a government committee is less likely to result in a sale than visiting a customer.

The challenge. For foreign information technology companies in Japan, the major challenge today is to prepare one's corporate family to handle the Japanese standards-setting access opportunities that exist.

Language. Understanding the Japanese standards process is not merely an exercise in sorting out the names of the various players, but is also one of dealing with the vast amount of information that they produce. For example, the Telecommunication Technology Committee's bound set of standards fills about three feet of space on a shelf. To understand how Japanese network standardization affects one's company, one may have to read through this mass of TTC information, plus another three feet of documents from other sources.

Not much of this material is available in English. MITI makes summary information available in English regarding the more mature proposed standards, but other ministries publish only in Japanese. There are few people in most American corporate families whose job it is to understand both Japanese and English and the Japanese standards process, and to serve as liaison between the Japanese standards process and the American technical standards process. Send he material to an outside translator, a competent one can be found, will typically cost 8,000 yen ($57) per page. A year's worth of standards can easily mount up to several thousand pages.

Not a job for salespeople. A number of American information technology companies in Japan have tried to use their sales engineers to participate in Japanese standards bodies. The results have usually been frustrating for both the participant and the company. A good salesperson wants to close a deal and move on. A good standards process participant has much more patience and concentrates on the details. Sales engineers will often have their performance in the company rated by sales. While contacts made in the standards-setting process can be useful in marketing, sitting on a government committee is less likely to result in a sale than visiting a customer. Much more satisfactory results have been obtained when the American company places a research and development-oriented employee on a Japanese standards body.

Short-term justifications. It is certainly difficult this fiscal year for an American company to demonstrate the financial benefit of budgeting a person in Japan to follow Japanese technical standards. This is not to say that the benefits are insignificant, but merely to say that they do not usually appear on a short-term financial statement. In some corporate cultures, the difficulty of quantifying technical or strategic considerations is understood; in others, it remains a source of contention.

Corporate support at home. Merely placing someone in Japan is not enough. That function must be supported not only in Japan, but also through liaisons in the US and perhaps Europe as well. The

timely transmission of business information worldwide is a skill that most Japanese manufacturers and many American service industries have mastered. I would not say that the same is true of American manufacturing in general, although at least one large, US computer company is, in internal information circulation, a match for anyone.

The outlook. Between 1984 and 1988, Japan's share of the worldwide data processing and office equipment market jumped from 14.8 percent to 27.8 percent, and Japan's share of the worldwide telecommunications market went from 14.4 percent to 21 percent. Japanese technology and the standards that support it are increasingly present around the world. Japanese technical standards will come to you, if not in Japan, then in the American home market. There is still time to prepare to meet them.

To go further

For a comprehensive view of Japanese information technology standardization bodies, see J. Muller and P. Kuhn, "Information Technology Standardization in Japan," Sept. 1988. Private printing available from TIT Consult K.K., Daita 6-6-14, Setagaya-ku, Tokyo, Japan 155, fax 81 (03) 475-6343

The author can be contacted as follows: John P. Stern, Vice President for Asian Operations, American Electronics Association, US Electronics Industry Japan Office, Yonbancho 11-4, Chiyoda-ku, Tokyo 102, Japan, phone (03) 237-7195, fax (03) 237-1237.

IEEE Standards Board okays revision work

The IEEE Standards Board has approved projects for revision of Standard C95.1, Standard Safety Levels with Respect to Human Exposure to Radio Frequency Electromagnetic Fields, 3 kHz to 300 GHz, and Standard C95.3, Recommended Practice for the Measurement of Potentially Hazardous Electromagnetic Fields — RF and Microwave. The board took action at its meeting September 28 and assigned the projects to IEEE Standards Coordinating Committee 28 (Nonlonizing Radiation).

The scope of these standards includes the biological effects of long-term exposure to low-level electromagnetic fields.

If you are interested in participating in the revision of these standards, contact John Woods, IEEE Standards Department, 445 Hoes Lane, PO Box 1331, Piscataway, NY 08855-1331.

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN 31261, SAUDI ARABIA

INFORMATION & COMPUTER SCIENCE DEPARTMENT

APPLICATIONS ARE INVITED FOR ASSOCIATE TO FULL PROFESSOR POSITIONS AS REGULAR FACULTY OR VISITING FACULTY. A. PH.D. IN COMPUTER SCIENCE AND EVIDENCE OF RESEARCH ACCOMPLISHMENT ARE REQUIRED. POSITIONS ARE ALSO AVAILABLE FOR LECTURERS WITH M.Sc. IN COMPUTER SCIENCE AND A MINIMUM OF TWO YEARS TEACHING EXPERIENCE.

FACULTY WILL INTERACT WITH UNDERGRADUATE AND GRADUATE PROGRAMS, AND WILL HAVE FREE ACCESS TO EXTENSIVE LAB, COMPUTER AND LIBRARY FACILITIES. THE COMPUTING FACILITIES INCLUDE IBM 3090, SEVERAL VAX 11/780, MICRO VAXES, INTERGRAPH, DEC & VAX WORKSTATIONS RUNNING ON ETHERNET AND OTHER SMALLER MACHINES.

KFUPM offers attractive salaries commensurate with qualifications and experience, and benefits that include free furnished airconditioned accommodation on campus, yearly repatriation tickets, ten months duty each year with two months vacation salary. Minimum regular contract for two years, renewable.

Interested applicants are requested to send their Curriculum Vitae with supporting information not later than one month from the date of this publication, to:

DEAN OF FACULTY AND PERSONNEL AFFAIRS
KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DEPT NO. 9023
DHAHRAN 31261, SAUDI ARABIA

Innovative people making technology work.

Varian Microwave Equipment Products is recognized as a world leader in international satellite communications, power amplifiers, transmitters and power supplies. Customers depend on the technologies we develop to maintain a competitive advantage in their respective industries.

Senior Computer Systems Engineer

You will have the lead position in continually upgrading our CAD/CAE capability. This will involve performing hardware/software evaluations and trade-offs for the acquisition, implementation and commission of this capability. Responsibilities also include interfacing the system with a Computer Integrated Manufacturing system for World Class operations as well as analyzing the automation potential of our products to improve reliability and cost-effectiveness.

A BSEE and 10 years' experience are required. Comprehensive knowledge of computer hardware/software technology and architectures plus fluency in several programming languages are essential.

Varian offers competitive salaries and a comprehensive benefits program designed to respond to your individual needs. Please forward your resume to: Varian Microwave Equipment Products, 3200 Patrick Henry Drive, M/S 15-38/CSE, Dept. C, Santa Clara, CA 95054. We are an equal opportunity employer.

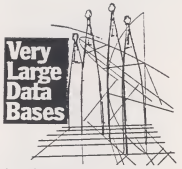
varian VA

CALL FOR PAPERS

VLDB'91

17th INTERNATIONAL CONFERENCE ON
VERY LARGE DATA BASES

SEPTEMBER 3-6, 1991
BARCELONA, (CATALONIA, SPAIN)



PAPERS

Original papers of up to 5.000 words in length are invited on topics including, but not limited to, those listed above.
All submitted papers will be read and carefully evaluated by the Program Committee. To submit papers, send five (5) copies of double-spaced manuscript, in English, including keywords, to one of the Program Co-Chairs by **FEBRUARY 15, 1991:**

Prof. Dr. Amilcar Sernadas
INESC
Rua Alves Redol, 9 ,7º.
Apartado 10105
P - 1017 LISBOA Codex (Portugal)
Internet (US) : inesc ! acs%solo@relay.EU.net
(Europe) acs%solo@inesc.uucp
or acs@inesc.ctt.pt

Dr. Guy M. Lohman
IBM Almaden Research Center
Dept. K55, Bldg. 801
650 Harry Road
SAN JOSE, CA 95120-6099 (U.S.A.)
Internet: lohman@ibm.com
BITnet: lohman@almaden

S. Abiteboul (France)
P. Apers (Netherlands)
F. Bancilhon (France)
D. Batory (TX, USA)
C. Beeri (Israel)
D. Bitton (CA, USA)
A. Borgida (NJ, USA)
F. Bry (Germany)
J. Bubenko (Sweden)
A. Buchmann (USA)
R. Carapuca (Portugal)
M. Carey (WI, USA)
M.A. Casanova (Brazil)
S. Chakravarthy (FL, USA)
S. Christodoulakis (CDN)
P. Dadam (Germany)
U. Dayal (MA, USA)
D. DeWitt (WI, USA)
K. Dittrich (Switzerland)
H. D. Ehrich (Germany)
E. Falkenberg (Netherlands)
J. L. Fiadeiro (U.K.)
J. Freytag (Germany)
H. Gallaire (France)
H. Garcia-Molina (NJ, USA)
G. Gardarin (France)
T. Haerder (Germany)
R. Hull (CA, USA)
Y. Ioannidis (WI, USA)
H. V. Jagadish (NJ, USA)
Y. Kambayashi (Japan)
R. King (CO, USA)
M. Kitsuregawa (Japan)
R. Krishnamurthy (CA, USA)
P. Larson (CDN)
B. Lindsay (CA, USA)
U. Lipeck (Germany)
F. H. Lochovsky (CDN)
P. Lockemann (Germany)
P. Loucopoulos (U.K.)
H. Lu (Singapore)
P. Lyngbaek (CA, USA)
R. Manthey (Germany)
Y. Masunaga (Japan)
D. McLeod (CA, USA)
R. Meersman (Netherlands)
C. Mohan (CA, USA)
M. Morgenstern (CA, USA)
E. B. Moss (MA, USA)
S. Navathe (FL, USA)
J. M. Nicolas (France)
S. Nishio (Japan)
A. Olivé (Spain)
F. Olken (CA, USA)
Z. M. Ozsoyoglu (OH, USA)
B. Pernici (Italy)
A. Pirotte (Belgium)
R. Ramakrishnan (WI, USA)
N. Roussopoulos (MD, USA)
R. Sacks-Davis (Australia)
G. Schlageter (Germany)
H. J. Schek (Switzerland)
J. Schmidt (Germany)
T. Sellis (MD, USA)
D. Shasha (NY, USA)
A. Solvberg (Norway)
R. Van de Riet (Netherlands)
Y. Vassiliou (Greece)
L. Vieille (France)
C. Zaniolo (TX, USA)

TUTORIALS

In cooperation with

IEEE COMPUTER SOCIETY

Proposals for tutorials should be sent by **January 6, 1991** to:

Prof. Antoni Olivé
Facultat d'Informàtica
Universitat Politècnica de Catalunya
Pau Gargallo, 5
E - 08028 - BARCELONA (Catalonia-Spain)
Ean: olive@lsi.upc.es
From BITnet: olive%lsi.upc.es.@cernvax

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC

PANELS

Proposals for panel sessions should be sent by **March 30, 1991** to:

Prof. Alfonso F. Cárdenas
University of California at Los Ángeles
Dept. of Computer Science
3731 Boelter Hall
LOS ANGELES, CA 90024 (U.S.A.)
Internet: cardenas@cs.ucla.edu

Computer Society NEWS

Editor: Guylaine M. Pollock, Sandia National Laboratories, Division 1424, PO Box 5800, Albuquerque, NM 87185; phone, (505) 846-0040; Internet, gmpollo@sandia.gov

NEWS FROM THE COMMITTEE ON PUBLIC POLICY

Identifying and mitigating risks to computer resources

Robert J. Melford, Member, COPP Subcommittee on Security and Privacy

Because society depends on the reliable operation of computer systems, it is critical that we understand their vulnerabilities. Publicity has focused attention on the activities of malicious outsiders. However, this accounts for less than 10 percent of all computer security losses. Employees are the predominant threat; their honest mistakes incur the single greatest loss while providing "on-the-job training" for those who are dishonest and disgruntled.

COPP's Subcommittee on Security and Privacy has been investigating computer-related crime issues. This report is a summary of the current study. Please direct comments and recommendations for further study to Glenn A. Myers, Subcommittee Chair, (319) 335-6134, e-mail gamyers@icaen.uiowa.edu.
—Ed.

What can go wrong. All computing systems can be compromised. One type of exposure occurs when a computer is precluded from functioning in accordance with its intended purpose. Examples include power outages and the Robert T. Morris, Jr., Internet "worm," which so overloaded afflicted computers as to essentially shut them down. This kind of event is most commonly referred to as denial of service, or the "availability" exposure.

Another type of exposure results when a computer's ability to operate as intended is partially impaired. This is described as a loss of integrity. The computing system may operate, but it no longer provides trustworthy results.

A third type of exposure exists because computing systems often serve as repositories of private, or otherwise confidential, information. Disclosure of that information to individuals not entitled to it can result in an invasion of privacy.

These three types of exposure occur through the very existence and use of computing systems, and risks can never be totally eliminated.

Computer crime. A loss incurred via any of these exposures results from either deliberate or unintentional acts, or from acts of nature. Deliberate acts often involve a crime. Unfortunately, there is no clear data identifying annual losses or the forms and distribution of computer crimes, because most of these crimes are never reported. Reasons vary from ignorance of the law to embarrassment. Sometimes the victim is unaware of existing laws encompassing a specific loss. Frequently, a legal remedy is not pursued because the victim has neglected to maintain the records needed to gather evidence.

Most often, however, the victim's own omission or commission — stemming from ignorance or negligence — provides the opportunity to commit the crime. As a result, the victim is more fearful of the costs associated with the potential public embarrassment from disclosure of the crime than of the losses incurred by the crime itself.

Identifying the threats. Estimates identifying the sources of all monetary losses stemming from security failings are in fundamental agreement. Here is the breakdown in approximate percentages:

Mistakes caused by honest employees account for approximately 65 percent of the aggregate financial loss. These errors of omission and commission are primarily attributable to stress, fatigue, poor training, and ineffective supervision.

Dishonest employees are responsible for an estimated 13 percent of all losses. The means for misappropriating cash or other financial assets are usually discovered through a mistake the employee makes that subsequently goes undetected by management.

Infrastructure failure contributes to about 8 percent of the losses. Interruptions of electrical, communications, and transportation services resulting from natural disasters or other external events are the most common examples.

Disgruntled employees are responsible for roughly 6 percent of the losses. As with the dishonest employee, the techniques used to trigger a loss are usually first discovered when management fails to recognize a mistake incurred by the employee.

Water causes about 5 percent of the aggregate damage. Computers are rarely the ignition source in a fire. Other equipment or material located within the facility that houses computers is usually to blame. Electronic equipment is damaged by the runoff of water used to extinguish the flames.

Losses due to outsiders, including most malicious "hackers," computer viruses, worms, and so forth, amount to a consensus figure of 3 percent. (Actual estimates range from 1 to 6 percent; the disagreement is due to the expense of chasing down rumors and false alarms stemming from the current public preoccupation with computer viruses and other malicious events.)

While the relative loss attributable to each category of risk has remained fairly constant, the total cost to individuals, organizations, and society from these losses has grown in parallel with the tremendous growth in the use of computers.

Using technology to improve security. Computer hardware and software

can be designed or upgraded to better resist security compromises. With hardware, for example, locks can be added to prevent theft. Backup power supplies can be used to protect against power outages. Microprocessor circuitry can be designed for decreased vulnerability to computer viruses.

Software can be designed to limit the possibility of mistakes or other problems. For example, answers can be restricted to a specific set of choices. Software can be used for cross-checking or maintaining an audit trail, thus identifying errors when they occur. Access to a computer system can be restricted to a specific set of users through software.

The privacy of data or information stored in a computer can be further protected through encryption — the same technique the US State Department uses to protect the confidentiality of diplomatic messages.

The prime limiting factors are cost and risk. The more complex and sophisticated the technical response, the greater the cost. And as complexity increases, so does the possibility that some new vulnerability will evolve from the very processes designed to decrease risk.

Mitigating computer security losses. While the risk of security losses can never be completely eliminated, their cost to the individual or organization can be reduced to an acceptable level. This can be accomplished by taking any of the following three measures, either singularly or collectively:

- Risk can be accepted. When the estimated cost of a loss is less than the expense of controlling it, the least costly option may be to do nothing.
- Risk can be assigned to others. Purchasing an insurance policy may be an economical alternative.
- Risk can be avoided by establishing security controls. The cost of using the control must include the estimated loss expected from the residual risk. These controls may be managerial or technical, or both.

The remainder of this discussion pertains to the third option.

Avoiding computer security risk. A program of risk avoidance consists of seven elements. Some deal with a risk before it develops, some are used when a problem actually occurs, and others are reserved to address problems that were not totally circumvented.

A policy establishing the mission of computer security within an organization and providing a source of broad guidance for its implementation and maintenance is the foundation of all security programs. The organization's chief executive officer (or equivalent) must encourage the development and activation of this policy. This is the foundation on which necessary standards, guidelines, procedures, and metrics will be built.

The second element of an avoidance program is prevention. For example, many mistakes can be avoided through adequate training. A denial-of-service exposure arising from the theft of a computer may be prevented by locking the door. Software controls can be implemented to restrict access to confidential information.

Detection and notification of an attempted or successful security compromise is the third program element. It has two components: First, there must be recognition of an attempted or actual breach of security; second, someone with enough authority and resources to respond to the detected activity must be informed of its occurrence.

A program must have the ability to mitigate, or limit, the impact of a security breach, along with the capacity to correct or recover from its effects. Thus, mitigation and correction constitute the fourth and fifth elements of a risk avoidance program.

Some security breaches result in the commission of a criminal act or cause a loss that may be recoverable through a civil action — or both. The sixth element, conviction, seeks to provide an organization with the necessary resources to pursue these objectives.

Finally, a security risk avoidance program is not just a plan that is implemented and subsequently forgotten. It is an ongoing project that draws on past experience to continually refine, improve, and update the program.

Conclusions and recommendations. Computer security is primarily an issue of sound managerial practices. Technological solutions can help, but these resources must be used within the context of a computer security strategy that management must set and enforce.

It would be unwise to disproportionately commit resources to problems that, while intriguing, are of relatively incidental importance. The greatest return on investment will come though a managerial approach to controlling employee errors. Computing systems can, at best, provide only a limited and secondary means to manage and govern the actions of those who use them.

The March 1990 IEEE-United States Activities' "Entity Statement on Computer Crime" recognized the need for any future legislation to focus on the proximate and subsequent consequences of acts committed instead of on the specific techniques by which malicious acts are committed. Legislation that is too narrow or too specific would soon be outdated.

Service awards presented at ICCD and ITC

Several IEEE Computer Society volunteers were honored in September at the IEEE International Conference on Computer Design (ICCD) in Cambridge, Massachusetts, and the International Test Conference (ITC) in Washington, DC.

At ICCD, Paul Horstmann received the Distinguished Service Award for "outstanding service to the IEEE ICCD conference as general chair and program chair in 1987 and 1986, respectively." William T. Dally received the Meritorious Service Award for "excellent support to ICCD by chairing its architecture track."

At ITC, Meritorious Service Awards were presented to

- Samuel Bedrosian, for "many years of dedicated service on the ITC Steering Committee."
- J. Reese Brown, for "many years of dedicated service on the Technical Committee on Test Technology's standards committees."
- Gerald Gordon, for "many years of dedicated service on the ITC Steering Committee and for serving as treasurer."
- Carnie Jackson, for "many years of dedicated service on the ITC Steering Committee and for acting as general chair in 1988."
- Wesley Radcliffe, for "many years of dedicated service to the VLSI Workshop, including four years as general chair."
- Rodham E. Tulloss and Colin Maunder, for "contributions and leadership as chair in the development and approval of IEEE Standard 1149.1, Boundary Scan Architecture."

Also at ITC, Certificates of Appreciation were awarded to H. Alan Hoover, Kenneth Mandl, Kenneth Parker, and James Tellier for "many years of dedicated service on the ITC Program Committee." M. Ray Mercer received a Certificate of Appreciation for "contributions as a member of the editorial board of *IEEE Design & Test.*

Gallager receives IEEE Medal of Honor

The IEEE presented its highest award, the IEEE Medal of Honor, to Robert G. Gallager, Fujitsu professor of electrical engineering and computer science at the Massachusetts Institute of Technology. The award, presented October 6 at the 1990 IEEE Honors Ceremonies in Toronto, Ontario, Canada, recognized Gallager for "fundamental contributions to communications coding techniques."

A member of the IEEE Computer Society and an IEEE fellow, Gallager has been at MIT since 1956 and is currently codirector of the Laboratory for Information and Decision Systems and cochair of Department Area 1 — Control, Communication, and Operations Research.

Gallager's early work on information theory focused on the reliability function of noisy channels. A resulting paper, "A Simple Derivation of the Coding Theorem and Some Applications" (*IEEE Transactions on Information Theory*, January 1965), won the 1966 IEEE Baker Prize and contributed to many later developments in coding. His book, *Information Theory and Reliable Communication* (Wiley, 1968), remains the standard graduate text and reference work for the field. More recently, his information theory work has focused on multiaccess communication.

Gallager was involved in founding Codex Corp. in 1962 and continues to consult for the company. His fundamental studies on quadrature amplitude modulation and detection led directly to development of the 9,600-bps modems that provided the company's commercial success. His inventions in coding and modulation led to four patents.

Robert G. Gallager

UPDATE

Contributions to Update are welcome. Send news of public policy or professional issues and of industrial or university research to *Computer*, 10662 Los Vaqueros Circle, PO Box 3014, Los Alamitos, CA 90720-1264, or to Editor-in-Chief Bruce D. Shriver, Vice President for Research, University of Southwestern Louisiana, Drawer 42370, Lafayette, LA, 70504.

New organization will manage, operate NSFnet

Merit, Inc., IBM, and MCI Communications announced the establishment of Advanced Network and Services, Inc., a nonprofit organization that will manage and operate the federally funded National Science Foundation Network backbone, under subcontract to Merit. Allan H. Weis, formerly with IBM, is the new organization's president and chief executive officer.

ANS has been established to help build and expand current networking capabilities to meet the heavy demand of the nation's scientists, engineers, and educators for high-speed networking. The company will provide networking services to researchers and educators in universities, federal laboratories, and the private sector.

IBM and MCI are providing ANS with initial funding, and Merit is providing network operations, network engineering and planning, and network information services.

Volunteers sought to help in Eastern Europe

The International Executive Service Corps is seeking retired computing managers and executives willing to go to Eastern European countries as volunteers to help companies become more effective and economically viable. The need is critical for skilled managers who can help companies in these countries function in a free-market economy.

Since 1964 the International Executive Service Corps, a nonprofit organization, has been sending volunteer executives and their spouses overseas to help in developing countries. They receive no salary, but they get travel and living expenses from IESC.

According to IESC, volunteers are motivated by their desire to put their knowledge and background to good purpose, an appreciation for new experiences and environments, and the satisfaction of knowing they are building better political and economic relations between the country they help and the United States.

For more information, contact IESC, 8 Stamford Forum, PO Box 10005, Stamford, CT 06904-2005; phone (203) 967-6000.

NRC merges computer, telecommunications boards

Recognizing the increasing linkage between computer science, telecommunications, and computer applications, the National Research Council has merged the Computer Science and Technology Board and the Board on Telecommunications and Computer Applications. The new, consolidated Computer Science and Telecommunications Board will continue with the current activities of the two separate boards. Its initial charge is to

- monitor and promote the health of the computer science, computing technology, and telecommunications fields;
- initiate studies involving computer science, computing technology, and telecommunications;
- respond to requests for advice from federal and private agencies and organizations on issues of computer and telecommunications systems planning, utilization, and modernization; and
- foster interaction between computer and telecommunication sciences and other pure and applied science and technology disciplines.

Joseph F. Traub, Edwin Howard Armstrong Professor in the Department of Computer Science at Columbia University, chairs the new board.

Editor: Richard Eckhouse, UMASS-Boston, Harbor Campus, Boston, MA 02125, Compmail ii, r.eckhouse, Bitnet, eckhouse@umbsky; Compuserve, 70516, 556

Object-oriented programming

T.L. (Frank) Pappas, AWEB Software Technology

Many people see object-oriented programming as the methodology of choice for writing reusable software. There doesn't seem to be a single definition of OOP that everybody agrees on, but it's most easily described as programming for simulation, based on modeling real-world objects within computer programs. In this review, I look at an implementation of Smalltalk-80, probably the best-known OOP language, and at a front-end processor for Ada that provides OOP support.

A Smalltalk primer. First, a brief look at Smalltalk itself. In Smalltalk, an object encapsulates private data and the operations, called *methods*, that can be performed on the data. A *message* is a request for an object to carry out one of its operations. The *receiver* (the object to which the message was sent) then determines how to carry out the operation. The set of operations to which an object can respond is called its *protocol*.

A *class* describes the implementation of a set of objects that have similar characteristics and attributes. The individual objects described by a class are called its *instances*. A class defines two kinds of variables: instance variables form an object's private data, and class variables are shared by all instances of the class. In Smalltalk, even a class can be treated as an object.

A class also defines two kinds of protocols. The class protocol is the set of messages to which a class responds; the methods comprising this protocol are called class methods. The instance protocol is the set of messages to which an instance responds; its methods are instance methods.

Figure 1a shows a Smalltalk class, called FinancialHistory, which uses only instance variables and instance methods. It was taken from *Smalltalk-80: The Language and Its Implementation* by Adele Goldberg and David Robson (Addison-Wesley, 1983). If this is the first time you have seen Smalltalk, you will probably find the syntax difficult to follow. You get used to it after using it for a short time.

The first line of Figure 1b creates an

```
class name                  FinancialHistory
instance variable names     CashOnHand
                            incomes
                            expenditures


instance methods

   transaction recording

      receive : amount from source
         incomes  at: source
                  put: (self totalReceivedFrom: source) + amount.
         cashOnHand ← cashOnHand + amount

      spend: amount for: reason
         expenditures  at: reason
                  put: (self totalSpentFor: reason) + amount.
            cashOnHand ← cashOnHand – amount

   inquiries

      cashOnHand
         ↑ cashOnHand

      totalReceivedFrom: source
         (incomes includesKey: source)
            ifTrue    [ ↑ incomes at: source]
            ifFalse:  [ ↑ 0]

      totalSpentFor: reason
         (expenditures includesKey: reason)
            ifTrue    [ ↑ expenditures at: reason]
            ifFalse:  [ ↑ 0]

initialization

   initialBalance: amount
      cashOnHand ← amount.
      incomes ← Dictionary new.
      expenditures ← Dictionary new

(a)

HouseholdFinances ← FinancialHistory new initialBalance: 350
BusinessFinances ← FinancialHistory new initialBalance: 5000
HouseholdFinances spend: 30.45 for: 'food'

(b)
```

Figure 1. A Smalltalk-80 class (a); using a class (b).

```
class name                  DeductibleHistory
superclass                  FinancialHistory
instance variable names     deductibleExpenditures


instance methods

   transaction recording

      spendDeductible: amount for: reason
         self spend: amount for: reason.
         deductibleExpenditures ← deductibleExpenditures + amount

      spend: amount for: reason deducting deductibleAmount
         self spend: amount for: reason.
         deductibleExpenditures ← deductibleExpenditures + amount

   inquiries

      totalDeductions
         ↑ deductibleExpenditures

initialization

   initialBalance: amount
      super initialBalance: amount
      deductibleExpenditures ← 0
```

Figure 2. A subclass.

instance of FinancialHistory, named HouseholdFinances, and gives it an initialBalance of $350. The second line creates a distinct instance, named BusinessFinances, with an initialBalance of $5,000. Each instance has its own copy of the instance variables. The third line sends the message "spend: 30.45 for: food" to the HouseholdFinances object.

An important OOP concept supported by Smalltalk is inheritance. Simply put, inheritance lets you specialize a class. For example, Figure 2 illustrates a new class, DeductibleHistory, for keeping track of tax deductible expenditures. This specialized class of FinancialHistory is called a subclass; FinancialHistory is a superclass of DeductibleHistory.

This means DeductibleHistory inherits FinancialHistory's instance variables and methods, that is, the subclass starts off with these variables and methods already defined. It specializes its superclass by adding an additional instance variable and three additional methods. In Figure 2, DeductibleHistory has overridden one of the inherited methods, initialBalance, providing a more appropriate definition for initializing the balance of a DeductibleHistory instance.

A class and its subclasses are all part of the class library. The ability to define classes and specialize them through inheritance is one of the major reasons that reusable software can be written so easily in OOP languages that support classes and inheritance. It is further aided by polymorphism, which allows different classes to have the same methods names. This is handled in Smalltalk by not binding messages to objects until runtime.

Objectworks

Objectworks is Parc Place Systems' implementation of Smalltalk-80. I evaluated version 5.2 on my Heath/Zenith AT/386 under Windows/286. You can also use Windows/386, but Windows 3.0 support won't be available until the next release. Objectworks requires 3 Mbytes of extended memory and at least 5 Mbytes of disk space. The MS-DOS version sells for $3,500. Objectworks is also available for Apollo, DEC, and Sun workstations and for the Macintosh.

Objectworks, like other Smalltalk environments, provides a host of features that greatly simplify production of Smalltalk software. Foremost is the integrated development environment, which lets you move easily from one tool to another. The tools include browsers, a source-level debugger, monitors for improving performance, and change management facilities. Objectworks also lets you include C code in Smalltalk programs, and it comes with a rich set of classes already available in the class library, including the source.

The browsers are especially useful. You can use them to view the source code for existing classes and methods, to modify existing methods and classes, to add new methods and classes to the class library, and to dynamically obtain cross reference information.

One thing I didn't like about Objectworks is its tutorial guide. Objectworks runs on many different platforms, but there is only one guide for all systems. Many file names are used throughout the tutorial, and since their names differ under each system, you must consult an appendix to determine the file names you should actually type in for your system. I really disliked that, and I hope it's changed in future releases.

Objectworks is highly portable. It has some special features for interfacing directly with the host operating system, but they don't seem to be needed, since any feature you need is already available. Parc Place claims that users don't have to change a line of code to move from Sun to Mac to DEC to PC. Unless you have some rare need to use one of the host system interfaces, that is probably true.

Other than the tutorial, I really liked Objectworks. If you are interested in OOP and are not restricted to a conventional language, you might want to try it. Even if you must use Ada, C++, Fortran, etc., consider Objectworks for rapid prototyping. Parc Place Systems is at 1550 Plymouth Street, Mountain View, CA 94043, phone (415) 691-6700.

Reader Service 21

Classic-Ada

Classic-Ada is a front-end processor that provides OOP facilities for Ada. I reviewed the Interactive 386/ix Unix version, which sells for $1,100. It's also available for Apollo, Data General, Sun, MAX/AUX, and DEC systems.

While many people talk about OOP and Ada as if Ada were an OOP language, that's really not true. Ada doesn't support inheritance, message passing, or dynamic binding.

As many of you know, Ada is undergoing one of the required periodic ANSI revisions, called Ada 9X. Some people are advocating that Ada 9X should include OOP support. Whether this will

```
WITH Tactical_Types; USE Tactical_Types;
CLASS Radar IS

   SUPERCLASS Fixed_Object;

   METHOD Create (New_Radar : OUT Object_ID);

   INSTANCE METHOD In_Range (Penetrator : Object_ID);
   INSTANCE METHOD Transmit_Message
         (Message : string; Destination : Object_ID);
   INSTANCE METHOD Out_Of_Range (Penetrator : Object_ID);

   INSTANCE METHOD Initialize;

END Radar;
```

Figure 3. A class specification in Classic-Ada.

```
CLASS Sam IS

   SUPERCLASS Radar;

   METHOD Create (New_Sam : OUT Object_ID);

   INSTANCE METHOD Add_Penetrator (A_Penetrator : Object_ID);
   INSTANCE METHOD Remove_Penetrator (A_Penetrator : Object_ID);
   INSTANCE METHOD Expect (This_Penetrator : Object_ID);

   INSTANCE METHOD Initialize;

END Sam;

WITH Tactical_Types; USE Tactical_Types;
WITH Penetrator_List; USE Penetrator_List;
CLASS body Sam IS

   Max_Shots            :  INSTANCE Integer;
   Possible_Penetrators :  INSTANCE Fighter_List;
   Alert_Delay          :  INSTANCE Integer;

   METHOD Create (New_Sam : OUT Object_ID) IS
   BEGIN
      New_Sam := instantiate;
      send (New_Sam, Initialize);
   END Create;

   INSTANCE METHOD Add_Penetrator (A_Penetrator : Object_ID) IS
   BEGIN
      Add_To_List (A_Penetrator, Possible_Penetrators);
   END Add_Penetrator;

   INSTANCE METHOD Remove_Penetrator (A_Penetrator : Object_ID) IS
   BEGIN
      Remove_From_List (A_Penetrator, Possible_Penetrators);
   END Remove_Penetrator;

   INSTANCE METHOD Expect (This_Penetrator : Object_ID) IS
   BEGIN
      Status := Alert;
      SEND (SELF, Add_Penetrator, A_Penetrator => This_Penetrator);
   END Expect;

   INSTANCE METHOD Initialize IS
   BEGIN
      Max_Shots := 4;
      Possible_Penetrators := null;
      Alert_Delay := 10;
      SEND (SUPER, Initialize);
   END Initialize;

END Sam;
```

Figure 4. Inheritance in Classic-Ada.

happen is not clear, although I'd certainly like to see it.

In the meantime, Classic-Ada offers a Smalltalk-like front end for Ada and provides classes as shown in Figures 3 and 4. These figures are simplified versions of examples from the Classic-Ada user manual.

Figure 3 illustrates a Radar class specification that is similar to an Ada package specification. "Class" is a reserved word here. A class specification can be compiled separately from its class body specification. In the Radar specification, an instance method is specified using the reserved words "instance" and "method"; a class method is specified without the word "instance." Methods have a form simliar to a procedure declaration. The reserved word "superclass" introduces the Radar superclass. Since Ada is a strongly typed language, a predefined name, Object_ID, denotes a class object in Classic-Ada.

Figure 4 contains the class specification and body for the Sam class, a subclass of Radar. As with Smalltalk, the subclass inherits the superclass' methods and instance variables. Sam also overrides the Create and Initialize methods.

The Radar class specification and the Sam class body illustrate that Classic-Ada lets you use separately compiled Ada units with classes, as indicated by the with-clauses and use-clauses that precede these units.

Instance variables are defined in a class body by adding the reserved word "instance" to an Ada variable declaration. Method bodies are similar to Ada subprograms.

In Sam, the methods Create, Expect, and Initialize each send messages using the Send primitive. In Smalltalk, the Send in Expect would be written as:

```
self Add_Penetrator : This_Penetrator
```

As in Smalltalk, the binding of messages to objects occurs at runtime.

The Classic-Ada preprocessor translates Classic-Ada code into pure Ada code. Class specifications and bodies are translated into package specifications and bodies, respectively. Methods are translated into Ada procedures. Classic-Ada syntax and semantics are checked by the preprocessor, but Ada code is only checked for syntax. The resulting translated program must be compiled by an Ada compiler.

The preprocessor also generates a Classic-Ada executive tailored to your Classic-Ada program. This executive, written in Ada, takes care of the dynamic binding of messages to objects. It must be compiled with your program. A nice feature of Classic-Ada is that the code it

produces seems to be portable in that it makes no assumptions about the host operating system or compiler. Thus, it seems you could develop a Classic-Ada program on a Unix platform and compile it under MS-DOS.

Classic-Ada generates a database to maintain class hierarchy consistency. This is analogous to the Ada program library's maintaining the consistency of separately compiled units.

Of course, since Classic-Ada is built on top of Ada, rather than as part of it, some things are a little awkward. For example, you must process Classic-Ada source through the preprocessor and then through a compiler, which can be a pain when debugging a program.

Also, class specifications allow Ada subprograms, variables, etc. to be declared. I wonder if this presents a consistent view of a class? (Why would a class provide both methods and subprograms?) On the other hand, a class specification cannot define a private part or be generic.

An area in which Classic-Ada fails miserably is the user manual. Frankly, it's one of the worst I've ever seen. For example, there isn't a complete Classic-Ada program in the entire manual. One of the three examples uses a tactical warfare simulation, which is where Figures 3 and 4 came from. The simulation is very large, so the manual presents only small pieces with no details about missing superclasses. I'm sure I don't have a complete understanding of how all the pieces go together. It shouldn't be difficult to come up with a well-organized manual with many examples and more detail about what is and is not allowed. The manual also needs more attention to typography and to the formatting of examples.

Aside from the manual, Classic-Ada is an interesting tool that is worth your consideration if you want to practice OOP in Ada. You can reach Software Productivity Solutions at PO Box 361697, Melbourne, FL 32936, or at 122 4th Ave., Indialantic, FL 32903, phone (407) 984-3370.

Reader Service 22

Hardware to support graphical user interfaces

Computing is aesthetics and power, in that order, and the manifestation of aesthetics these days is the display of graphical user interfaces (GUI). To really appreciate some of the new GUIs and their associated new generation of applications, nothing is more important than a high-quality monitor and graphics adaptor board. Seiko's noninterlaced, multisynching monitors, using Sony Trinitron CRTs, are high-quality displays that can provide the best interface that GUI-based applications have to offer.

The 14-inch Seiko CM-1450 ($999) and its 20-inch big brother, the Seiko CM-2050 ($2,999), can turn hours of computer use into a pleasurable time that's easy on the eyes. With horizontal and vertical scanning frequencies ranging, respectively, from 31-50 KHz and from 50-90 KHz, dot pitches of .25 mm and .31 mm, and a notable absence of flicker, the Seiko nonglare silica-coated screens provide clear and crisp renditions of software application interfaces.

I tested both monitors with the Sigma VGA Legend graphics adapter using software drivers for Windows 3.0 and with resolutions ranging from CGA to noninterlaced 1024×768 pixels with 256 colors. While the display of the 14-inch monitor was very readable at high resolutions, the 20-inch version was much easier on my middle-aged eyes. Of course, the larger screen size has related disadvantages. The CM-1450 is a "conventional " size (33 pounds) and can be placed on any desktop or monitor stand, whereas the CM-2050 weighs in at a hefty 75 pounds with a 13-square-inch footprint. Both products have a tilt-and-swivel base (which is detachable on the CM-1450), a power cable, and a 15-pin male D-video connector cable. The cables are built into the CM-1450 and detachable from the CM-2050. In addition to the female D-type video connector on the CM-2050, there are also BNC connectors for separate RGB and synch inputs as found on higher performance graphics adaptors.

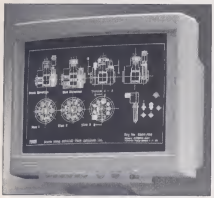
Except for their sizes, both monitors are set up similarly. The on/off, brightness, and contrast controls are on the right side of the CM-1450, with horizontal, vertical centering, and size controls somewhat inconveniently placed on the back. However, once I had set these controls I never really needed to readjust them, even as the displays automatically locked into the different display modes offered by the Sigma Legend's utility software. On the CM-2050, all the controls are on the front, together with vertical and horizontal static convergence controls whose effects on displayed images seemed negligible.

Inevitably, comments on the merits of a display are subjective. I used both monitors extensively at the highest resolutions with "serious" applications that included Microsoft Word for Windows, the Windows version of Lotus 1-2-3/3 (using Iris's 3-For-3), and a host of windowable DOS applications. But the monitors' finest moments came when I ran an adventure game called Indy, The Graphics Adventure. This program by Lucasfilm Games is essentially a VGA-level, animated version of the movie *Indiana Jones and the Last Crusade* with integrated user functions. The animation sequences on the CM-2050 actually gave me some insight into the potential of HDTV — these monitors are clearly indicative of the emerging technologies that are precursors to integrated computer and digital television products.

Sigma VGA Legend. The Sigma VGA Legend display adaptor is a 16-bit board that operates with the full range of graphic display standards. The 512-Kbyte version ($449) supports the following display specifications: 640×480 pixels with 256 colors, 800×600 pixels with 16 and 256 colors, and 132-column text mode. If you purchase the board with 1 Mbyte of memory ($569), you gain the highest resolution — 1024×768 pixels with 256 colors. (A user-installable set of four RAM chips is available separately if you want to upgrade the 512-byte version to 1 Mbyte.) Of course, your display must have appropriate specifications to get the higher resolution output.

Sigma says this board has a VESA-compatible BIOS, but I wasn't able to test that. What I did find was that this is the first board I've tested that supports multisynching monitors from 56-88 Hz in text mode and the new VESA standard of 72 Hz in enhanced graphics modes.

The board comes with drivers for Windows/286, /386, and 3.0, Lotus 2.x and 3.0, Wordstar 3.3 and 5.0, Wordperfect 5.0, Autoshade and Autoshade 386, GEM 3.x, Ventura 2.x and 3.x, Microsoft Word 5.0, AutoCAD 9, 10, and 386, and Generic AutoCAD. Sigma also expects to ship a driver for Presentation Manager soon.



All the controls on Seiko's CM-2050 are on the front. The 20-inch, 75-pound display has a 13-square-inch footprint.

I first installed the board in a 16-bit slot of a Toshiba T3200 and was unable to boot the computer. Because the board's physical size precluded testing it in the machine's 8-bit slot, I tried another version of the board, still with no luck. A technical support person at Sigma promised to solve the problem, but when he called back he said Toshiba was unable to duplicate the problem with the T3200s they are now shipping, and that Toshiba would contact me to resolve the incompatibility of the two systems.

I made the board work in a 16-bit slot of an AST 386/C, but somewhat idiosyncratically. The board is designed to operate in plain vanilla VGA right out of the box. I tested it in this mode, first with an IBM VGA display and then with the Seiko monitors. In all cases, I could use the plain VGA mode as well as the special installation software to set various display modes to operate in DOS. The setup software creates a file called monitor.com whose contents depend on the graphics mode and display you select.

When I used the Microsoft-supplied VGA driver or any of Sigma's higher resolution drivers on the 386/C, I ran into trouble with the IBM and the Seiko monitors. Whenever I did a cold boot and tried to run Windows 3.0, the machine hung up. If I then did a warm boot and tried to run Windows, the program and its display adapters worked fine. All things being equal, I decided to pull the board and replace it with another VGA board to see if the problems were repeatable. They were not, which suggests an incompatibility between the Sigma board and the AST machine's ROM BIOS. I didn't mind the work-around solution, though, because of the resulting high resolution of the display system.

In another test with a 33-MHz PC Genius 386 and an old NEC multisynch monitor, everything worked fine in both 640×480 and 800×600 resolutions. The higher resolution demanded a bit more than the old NEC monitor was initially up to — I had to let it warm up for 15 minutes before it synched up. But I mention this only for those of you who don't have the newer NEC Multisynch 2A that Sigma recommends. With a little luck you can expect the older monitor with its more limited range to work.

Another point worth mentioning is that I found I didn't like Sigma's parameters for the 800×600 display mode, so I just edited the setup.inf file in the windows subdirectory to set the character size to 100×96×96. The results were much more pleasing, and I suggest you consider this change as well.

The board comes with one 5.25-inch setup disk and two disks of software drivers, a 49-page user guide describing installation and setup, and a screen-saver program. Although the board's performance was very good, Sigma includes a utility called Biospeed to improve it. A second, 53-page software support guide describes the installation and use of the various application drivers. Instructions on recent additions to the drivers list (such as Windows 3.0) are included on extra sheets.

Both the displays and the VGA adapter board come with a one-year parts-and-labor warranty. All in all, I found the products quite satisfactory despite the incompatibilities. I recommend this combination of the Sigma Designs board and one of the Seiko displays, but I do suggest that you test the Sigma board in your computer before you commit to it.

You can contact the PC Products Division of Seiko Instruments USA at 1144 Ringwood Ct., San Jose, CA 95131, phone (408) 922-5900. Sigma Designs is at 46501 Landing Pky., Fremont, CA 94538, phone (415) 770-0100.

—*S. Reisman*

Seiko monitors: **Reader Service 23**
Sigma board: **Reader Service 24**

Microsoft C 6.0

Microsoft C 6.0, the first major upgrade of the popular compiler in about two years, includes a number of enhancements and new features, including an integrated development environment. Microsoft apparently finds the new features significant enough to refer to the new version as a "professional development system" rather than simply as a compiler with associated tools.

The new system comes with both 3.5-inch and 5.25-inch disks, a welcome change from previous versions. The installation process is very similar to that of version 5.1 and is relatively painless. Disk space used can run anywhere from 6 to 10 Mbytes, depending on the options you select during installation.

Two separate compilers are available with Microsoft C 6.0: a full compiler with extensive optimization features, and a quick compiler that is much faster than the full compiler but cannot perform the full set of optimizations. The quick compiler is ideal for the development phase, and the full compiler can be used when your program is debugged and ready for release. If you use the quick compiler you also have access to the incremental compiler, which in turn lets you use the incremental linker. The incremental compiler only recompiles functions that have changed since the last time you compiled, a big advantage during development. The incremental linker can substantially speed up the linking process by only relinking modules that have changed since the last link. However, to use the incremental linker, you first must perform a full link with the /inc incremental command-line option. Whenever a full link is subsequently required, the incremental linker will invoke the full linker. Quick compilation, incremental compilation, and incremental linking are all available from either the Programmer's Workbench or the system command line.

Microsoft has added a number of new keywords to the language to increase program performance and programmer productivity. _fastcall instructs the compiler to place function arguments in registers whenever possible, rather than pushing them onto the stack. If the appropriate registers are not available, then the stack is used. This option can be applied at the module level by using the /gr compilation flag. If the option is selected, the arguments of all functions in the module will be passed in registers unless specifically overridden. If you do not use the compiler command-line option, indi-

vidual functions can be declared with _fastcall.

_asm invokes the in-line assembler and can appear anywhere a C statement is legal. The in-line assembler can accept any expression that is valid in the Microsoft Macro Assembler, including all 80286 and 80287 instructions. C variables visible in the block where the _asm keyword appears can also be used in an assembly language instruction. As an example, the instruction "mov ax, item" is valid if item is visible.

The syntax of _based is "_based(base) <declarator>." It can be used much as you would use either _near or _far. _based specifies that a data object will reside in the segment specified by "base" and not in the current data segment. One of this keyword's major advantages is that based pointers occupy only 16 bits and are added to the base to form a 32-bit pointer. Finally, _segment is a primitive data type used to store a segment address. It can be used in conjunction with _based to declare a based variable.

Microsoft C 6.0 has a number of new compiler command-line options. A tiny memory model is included and can be invoked with the /at option, and the /w4 option activates a new Lint-style warning level. A number of other options deal with new optimization features such as enabling global register optimization, turning on global optimization and global common subexpressions, and enabling aggressive optimization. Optimization levels can also be controlled from the function level via pragmas.

The new version also includes the newest revision of the Codeview symbolic debugger. If you are hesitant about upgrading to the new system, the improvements to CodeView alone should convince you. There are many substantial improvements, the most significant of which is the ability to deal with much larger programs and many more symbols. If you have an 286- or 386-based machine with at least 384 Kbytes of extended memory, then Codeview 3.0 will need less than 12 Kbytes of conventional memory while running from the extended memory. I used the new Codeview with two programs I had been unable to debug because of their large size — approximately 500 Kbytes each. (In fact, one of the programs could not even be run with Magic CV, another product that uses a 386 and the conventional Codeview to run from extended memory.) In both cases, I was able to load and execute the programs with no problems.

Codeview 3.0 now has a built-in command-line editor, eliminating the need to constantly retype long, complicated commands. Codeview will also save the state of your debugging session when you exit, and it will restore the state the next time the session is loaded. It will save all previously set breakpoints as well as the window environment. These features can greatly improve your productivity and should have been available long ago.

Another important enhancement in Codeview is the separate window for viewing local variables. The window lists the values of all variables local to the current function or block. A second window for viewing source code or include files has also been added. You can view the same source file in the second window if you choose. One window can display the C source code, while the second can display the assembly code associated with the source file. New windows have also been added for viewing the status of a 8087, 80287, or 80387 coprocessor in your system and for viewing specific areas of memory during program execution. You can display all of these windows simultaneously.

Previous versions of Codeview have not always been pleasant to deal with. Codeview 3.0 is a major improvement and should become a mainstay.

Previous versions of Codeview have not always been pleasant to deal with and were often very temperamental. Codeview 3.0 is a major improvement and should become a mainstay.

Unfortunately, the Programmer's Workbench is a major disappointment. Its biggest problem is speed, especially the time to load itself into memory. This is most noticeable if you are working under DOS. In OS/2, you can create a session to run the workbench, and then toggle in and out without reloading. But if you ever need to leave and reenter the workbench under DOS, be ready for a long delay. I used a 20-MHz 80386-based machine and still found the workbench frustratingly slow.

The hub of the workbench's design is the programmer's editor. It is very similar to previous versions of the Microsoft editor but with many new features. It is completely configurable, but its methods for modifying the configuration are clumsy and often use cryptic names for various system options. There are two configuration methods: you can use the Editor Settings selection from the Options menu, or you can edit the tools.ini file directly. Changes made through the Options menu apply only to the current session unless you specifically save them in tools.ini, while changes made to tools.ini are always loaded along with the workbench. Compared with the methods that other programmer's editors offer for changing configurations, Microsoft's methods are antiquated.

The workbench editor handles several other common functions poorly. The blocking of text is cumbersome. There are three separate text-selection modes: box, line, and stream. You change the current mode by toggling through a list of available modes either with the mouse or through the Edit menu. It is not always clear in the list which is the current mode and which is the next mode. If you are accustomed to selecting modes directly, you will find this method uncomfortable.

If you want to use your own editor instead of the workbench editor, you can add it to the Run menu list. I tried this approach, but the time to exit the workbench and then return from the optional editor when finished was unacceptable. Under OS/2 you can run your own editor as a separate session, improving the speed problem, but this solution is of no use under DOS.

The Programmer's Workbench includes a built-in feature for browsing through program variables and functions and for exploring the program structure. To use the Browse option you must compile and link your program with the Browse option enabled. This builds a database of information on disk that the browser can interrogate. Once the database is available, you can perform a number of interesting functions with the browser. You can list all variables and symbols appearing in either program modules or include files, and you can view an item's point of definition. You can also display a list of all points in your program where a particular variable or symbol is used. The browser will also display a tree structure of function calls within your program. I found the ability to go directly to a program variable's point of definition to be the Browser's most useful feature. It proved a valuable supplement to Codeview during debugging.

The Programmer's Workbench provides a very convenient facility for building and maintaining programs with a large number of modules. It can automatically create Make files for both debugging and release versions of your program. You can also use it to easily control the list of modules comprising

your program. The list can include source modules, object modules, and even library files. There is also a list of predefined options for compiling and linking your software in a number of different environments, such as DOS, OS/2, and OS/2 Presentation Manager, with or without multithreaded functions.

Once you have selected a predefined configuration, you can change any of the compiler or linker options and save the configuration for use with other Make files. After choosing or building a Make file, you can rebuild the entire program or only the portions that have changed since the last rebuild. If compile or link errors occur during the build process, a window can display them. Any compile errors are referenced to the appropriate source file, which is displayed in the source window. If the program is built successfully, you can run it directly from the Programmer's Workbench or invoke Codeview for debugging.

The printed documentation of Microsoft C 6.0 is minimal, at best. It consists of a user manual for the Programmer's Workbench, a C reference manual, and a manual covering advanced programming techniques. The quality and usefulness of the manuals is questionable. The workbench manual is only about 75 pages and amounts to little more than an installation handbook and basic guide to familiarize users. The C manual is aimed at experienced professionals and would be of little use to novice programmers, since it consists mostly of tables and lists of program options. The manual covers library functions, but none of the references show how to use them. Microsoft recommends that you buy a copy of the *Microsoft C Runtime Library Reference* (Microsoft Press), but it would have been better to include that book with the documentation. The manual of advanced programming techniques covers all components of the Programmer's Workbench in detail, but not as much detail as the C manual. It covers subjects such as program optimization, customizing the workbench, managing memory, and using the Make facilities.

Microsoft relies heavily on the On-Line Advisor to supplement the documentation. The advisor is available through either the Programmer's Workbench or a stand-alone Quick Help program. In both cases the information is complete and well organized. It covers every aspect of the Programmer's Workbench and associated tools, including the C library, Codeview, error codes, C syntax, and the Programmer's Workbench itself. The help facility for the C library includes an example of each function implementation, which you can cut and paste directly into your code. You can also print any of the help topics and create your own documentation package.

The Programmer's Workbench implementation of the On-Line Advisor is tightly integrated with the Microsoft editor. When your program appears in the source window, you can click on any keyword with the mouse and retrieve the help associated with that keyword. The same approach applies to error codes in the compilation error window. In fact, you can click on any word known to the advisor and get the associated help text. The On-Line Advisor provides true hypertext capabilities, letting you use links to traverse a set of related topics.

The Programmer's Workbench is a major disappointment. Its biggest problem is speed. I used a 20-MHz 386 and still found it frustratingly slow.

Although I found much of the printed documentation disappointing, the On-Line Advisor is a valuable asset.

I found a number of problems while using Microsoft C 6.0 on a large, complex system consisting of many modules, include files, and external libraries. The system had been developed with Microsoft C 5.1 using simple batch files to maintain control of the compile and link process. It had been fully debugged and was in working order. I first tried to compile and rebuild the system with the DOS version of the Programmer's Workbench. The quick compiler reported a compiler stack overflow when I tried to compile one of the smaller modules, and it claimed the program was too complicated. When I disabled the quick compiler, the full compiler was able to process the program. When I repeated the same process under OS/2 with the quick compiler, no such problems occurred.

The linker also caused some problems under DOS. It consistently ran out of symbol space, although the version 5.1 linker was able to handle the task. Again, I had no such problems under OS/2. Use of the optimization features proved to be a major problem. The situation became so bad I was often forced to abandon any attempt to use these features. Microsoft has a long way to go in this area. There were other minor problems with version 6.0, but these were the most serious. For smaller, less complex systems, I did not experience any similar problems.

The improved Codeview debugger and the On-Line Advisor make the upgrade cost worthwhile. The Programmer's Workbench, however, leaves much to be desired. It is far too slow in the DOS environment, and the quick compiler has some serious problems with very large and complex programs. The editor, Make facility, and browser are all disappointing and clearly not as sophisticated as other similar products. I hope the workbench's open architecture will improve the situation soon.

If you do not currently own Microsoft C, you would be advised to look elsewhere or to wait until Microsoft corrects their problems. Microsoft C 6.0 is available from Microsoft Corporation, 1 Microsoft Way, Redmond, WA 98052, (206) 882-8080. It sells for $495.

—*D. McAuliffe*

Reader Service 25

A combination mouse, trackball, and keypad

My first impression of the Trackermouse Plus from Penny & Giles Controls was that it was a bad idea, poorly implemented. My initial opinion was reinforced as I moved from observation to use.

This $250 device consists of a 2.5×7 inch plastic case that tapers up from 1.5 inches to .5 inch, with a 1.5-inch phenolic ball exposed at the thicker end as a trackball on the top and as a mouse on the bottom. The top of the case has a keypad with 12 keys arranged in a standard touch-tone telephone pattern. Two buttons, one on each side of the case, complete the controls. The three-foot connecting cord terminates in a nine-pin D-type connector to attach to a serial port, either directly or using the 9-to-25-pin adapter provided.

The Trackermouse Plus comes with a 17-page manual and a 360-Kbyte disk. The documentation is barely adequate as an installation guide, and it provides little help toward understanding and using the product.

Setting up the Trackerball involves installing and configuring a mouse driver and a keyball driver. P&G says any Microsoft-compatible mouse driver can be used, but be careful using the P&G

driver. Microsoft has updated its mouse driver with each new software release, and older versions of even its own mouse driver won't work with the latest software. For example, the P&G mouse driver would not work correctly with the latest version of Microsoft Works, so I had to use the Microsoft driver that came with that software.

Once you have installed a mouse driver, you can use the Trackerball as a simple one-button pointing device. You rotate the ball either by turning it with a finger (the trackball method) or by tilting the case so that the ball rolls on the table (the mouse method). However, using the Trackermouse as a mouse is folly; you'll need exceptional manual dexterity and tolerance to muscle cramps to keep the ball on the table and to move the device without accidentally pressing the side buttons.

The Trackermouse's virtue presumably lies in its 12-button keypad, which is supposed to translate its control inputs into equivalent keyboard commands, letting you use the product with programs that do not support pointing devices. You configure the keys with keyball.com, which you can run interactively or provide arguments for at the command line. In either case, you determine which pointing device outputs to translate or ignore, cursor precision and speed, linear locking (diagonally, vertically, or horizontally), which keyboard keys correspond to up, down, left, or right pointer motion, which two keyboard keys correspond to the Trackermouse buttons, and the sequence of keyboard keys that the Trackermouse keys generates.

Keyball.com is an annoying little program. In interactive mode, it starts by executing a P&G color advertisement (which takes 112 Kbytes of disk space) and then takes you through eight screens to configure the keys. With care and luck, you can work through the program without rebooting the system to produce a named configuration disk file and update the keyball driver. The keyboard key definitions appear on the setup screens as four-digit scan codes — a real frustration when reading or editing an existing configuration. After 10 tries and nine reboots, I ran through the entire keyball program and created a configuration file. I was keen to see how the definitions assigned to the keypad would work. They didn't.

Given the shamefully bad software, inaccurate and incomplete documentation, and poor ergonomic design, I cannot recommend this device. Penny & Giles Controls is at 35 Reynolds St., Attleboro, MA 02703. — *D. Somerville*

Reader Service 26

UNIVERSITY OF
St. Thomas

DEAN, GRADUATE SCHOOL OF TECHNOLOGY

The University of St. Thomas, a Catholic comprehensive university committed to the liberal arts and community service, seeks an experienced individual to serve as Dean of its newly created Graduate School of Technology. St. Thomas enrolls approximately 5000 undergraduate students and 4000 students in professional graduate programs, including approximately 300 and 250 students respectively in the two applications oriented degree programs currently constituting the graduate school — a master's in Software Design and Development and one in manufacturing Systems Engineering.

We seek an individual to:

- take primary responsibility for the public relations and development activities of the school (in conjunction with central University offices).
- work collaboratively with the chairs of the manufacturing Systems Engineering and Computer Science departments of the school on the preparation and approval of budgets, the evaluation of faculty (including recommendations concerning rank and tenure), the hiring of (including recommendations concerning rank and tenure), the hiring of new faculty and the long range plan for the school.
- chair the graduate school's curriculum committee.

The qualifications sought in applicants for this position are:

- an earned doctorate, preferable in engineering or computer science.
- ability to help new graduate programs and prospective undergraduate engineering programs in general engineering and manufacturing/industrial engineering programs in general engineering and manufacturing/industrial engineering achieve ABET accreditation.
- the ability to nurture and support the master's program in Software Design and Development and a prospective doctoral program in the same discipline.
- knowledge of the conditions of contemporary industry and experience as a cost effective manager are required.
- work experience in industry is strongly preferred.
- success in fund raising, grant acquisition and public relations activities preferred.
- commitment to the goals of the University of St. Thomas.

This position will include a ranked faculty appointment. Starting date is negotiable.

The deadline for applications and nominations is November 19, 1990. Applications should include a letter of interest, a curriculum vitae and the names, addresses and telephone numbers (not letters) of three professional references. Applications should be sent to, Dr. James F. Reid, University of St. Thomas, 2115 Summit Avenue, Mail #4059, St. Paul, Minnesota 55105.

The University of St. Thomas is an equal opportunity/affirmative action employer.

Moving?

Name (Please Print)

New Address

PLEASE NOTIFY US 4 WEEKS IN ADVANCE

City State/Country Zip

MAIL TO:
IEEE Service Center
445 Hoes Lane
Piscataway, NJ 08854

ATTACH LABEL HERE

- This notice of address change will apply to all IEEE publications to which you subscribe.
- List new address above.
- If you have a question about your subscription, place label here and clip this form to your letter.

NEW PRODUCTS

Contact or send releases to Christine Miller, *Computer*, 10662 Los Vaqueros Circle, PO Box 3014, Los Alamitos, CA 90720-1264; Compmail II, c.miller

XMS minisupercomputer eases entry to Cray line

Cray Research's XMS system is a 64-bit minisupercomputer based on the company's vector architecture and compatible with the Cray X-MP system.

The company says the new system can serve as an application-development platform for Cray systems, as a department-level system compatible with Cray systems elsewhere in an organization, and as a Cray-compatible platform for secure or remote operations.

The XMS has peak vector performance of 36 Mflops and peak scalar performance of 18 Mflops. It has one central processing unit and is available with 32, 64, or 128 million bytes of memory.

The air-cooled XMS runs Cray's Unicos operating system, which is based on Unix System V and is used throughout the Cray line. The XMS also runs the same compiler software as the larger Cray systems, including the CF77 Fortran compiling system and Standard C, along with a variety of library routines and utilities.

Reader Service 30

Apple launches new Mac attack

Apple has announced three lower priced Macintosh systems and several new peripherals.

The Macintosh Classic replaces the Macintosh SE and features the same "all-in-one" design. The machine is based on Motorola's 68000 microprocessor. A Macintosh Classic with 1 Mbyte of RAM and a Superdrive floppy disk drive costs $999. With 2 Mbytes of RAM and a 40-Mbyte hard disk, the system retails for $1,499. It is available worldwide immediately.

The color Macintosh LC features a new modular design and includes built-in support for three monitors and up to 256 colors. Based on a 16-MHz Motorola 68020 chip, the LC costs $2,499 with 2 Mbytes of RAM and a 40-Mbyte hard disk. It will be available in limited quantities in December, and will be available worldwide in the first quarter of 1991.

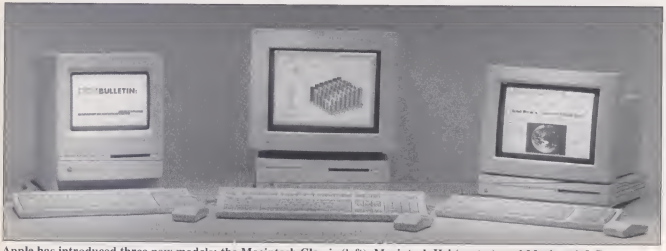
The Macintosh IIsi becomes the least expensive Macintosh II while reportedly equaling the performance of the Macintosh IIcx. The system — based on a 20-MHz Motorola 68030 — provides built-in support for four different monitors and up to 256 colors or gray shades. A base system with 2 Mbytes of RAM and a 40-Mbyte hard disk costs $3,769. With 5 Mbytes of RAM and an 80-Mbyte hard disk, the system retails for $4,569. It is available worldwide immediately.

The Macintosh Classic and the Macintosh LC include a keyboard, while the IIsi separately offers a choice of keyboards. All the systems come with a mouse, system software, and Hypercard 2.0.

Also, both the LC and the IIsi have sound input capabilities and come with a microphone that lets users record and store voices. The LC will also be able to use a $199 expansion card, called the Apple IIe Card, that lets the machine run applications written for the Apple II. The card, which will be available in March 1991, will also support joysticks and 5.25-inch floppy disk drives.

Apple has also announced two 12-inch displays, one RGB and one monochrome. The color monitor retails for $599, and the monochrome monitor costs $299. Apple's current High-Resolution Monochrome Monitor remains available at a reduced price of $299.

Macintosh Classic: Reader Service 31
Macintosh LC: Reader Service 32
Macintosh IIsi: Reader Service 33
Apple IIe Card: Reader Service 34
RGB monitor: Reader Service 35
Monochrome monitor: Reader Service 36



Apple has introduced three new models: the Macintosh Classic (left), Macintosh IIsi (center), and Macintosh LC.

Xerox' Globalview environment embraces new software, hardware products

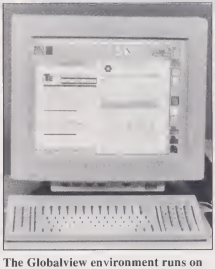
New hardware and software products are part of Xerox' new Globalview network computing environment. Xerox says the system is the foundation for a range of applications, including integrating text, graphics and data into compound documents; managing projects and teamwork; and accessing and sharing documents across networks.

Applications built on Globalview have a consistent, icon-based graphical user interface. They support more than 100 foreign languages and dialects, graphics capabilities, and transparent data access on host or departmental computers, according to Xerox.

The new Xerox 6520 workstation, based on Sparc/Unix technology from Sun Microsystems, runs the Unix version of Globalview. The workstation operates at 15.8 MIPS, comes with 8 Mbytes of memory (expandable to 48 Mbytes), and includes Sun OS, Sun's standard Unix operating system. Globalview starter kit software for the 6520 workstation starts at $9,995.

Xerox also announced Globalview for the PC, which consists of a coprocessor board that plugs into a standard AT bus, plus a 19-inch high-resolution display. The product turns DOS and OS/2 computers into multitasking, networked workstations, according to Xerox. Globalview for the PC starts at $3,995, including starter kit software and a 19-inch monochrome monitor.

Xerox' Docuteam software for the Globalview system focuses on the document as the primary means by which business teams develop and exchange information. Docuteam organizes documents into shared electronic libraries where users check material in and out. As a document is checked into the library, Docuteam can extract structured information such as graphs, tables or relational data from the document and stores it in an Oracle relational database running on DEC VAX system. Docuteam prices are based on the number of desktops licensed, beginning at $20,000 for 10 desktops. The price includes software for 10 desktops and software to integrate the Oracle database into the network.



The Globalview environment runs on Xerox' 6520 Sparc-based workstation.

Globalview products combining software and hardware include Docubuild, and Docutech. Docubuild is designed for workgroup publishing, especially medium and large multiauthor documents such as product manuals, technical proposals, scientific reports, and regulatory filings. The system includes patented high-speed pagination technology and WYSIWYG support for compound Standard Generalized Markup Language documents. Docubuild lets multiple authors in different locations access and modify documents across networks, manage work flow, and simultaneously view the status of all aspects of a publishing project. Docubuild will begin shipping in the second quarter of 1991. Prices start at $2,500 per workstation and $495 per PC.

Docutech Production Publisher provides high-speed, high-resolution digital image capture and document manipulation capabilities. It produces up to 135 prints per minute and incorporates a scanner that creates 600-dpi digital masters from hard-copy originals for storage, retrieval manipulation, and printing. The system also includes a 600-dpi laser printer with in-line collating, stitching, binding and stacking capabilities. The list price is $220,000, and rental, lease, and purchase plans are available.

Reader Service 37

Intel aims for notebook computer market with new products

Intel has announced several new products aimed at the notebook computer market, including the 386 SL microprocessor and 82360SL I/O subsystem chip. Intel claims that the two chips, plus peripherals, allow a complete 386 PC AT to be built on a 4×6 inch board.

The 386 SL is a single VLSI device integrating a 20-MHz, static 386 core with the system's cache controller, main-memory controller, LIM 4.0 EMS mapping logic, PC AT and ISA bus interface, 387 SX coprocessor interface, and power conservation logic. The chip also implements the system management mode.

The 82360SL chip provides all the serial, parallel, DMA control, memory refresh, real-time clock, chip-select, timer/counter, interrupt control, and hard-disk interface functions. The chip also includes a programmable power management subsystem that lets system firmware direct where and when power is consumed.

The 386 SL is available in a 227-pin land grid array. The 386 SL CPU and the 82360SL chip are available in a 196-lead plastic quad flat pack. In quantities of 1,000, the 386 SL is $176 and the 82360SL is $45.

Other products announced include the 80C51SL low-power notebook keyboard controller, the 89C024FT low-power modem chip set (which reputedly allows a complete modem to be built in less than three square inches), and several flash memory products.

The Flash Memory Card is a memory card based on flash memory technology. Flash memory is nonvolatile (it retains data when the power is turned off).

Intel says the credit card-sized package increases a portable PC's performance and durability, while reducing its power consumption (compared with traditional mechanical disk drives).

Intel claims that portable PCs using flash memory could see a 10- to 100-fold increase in battery life over portable systems using mechanical memory devices, due to the low power needs of solid-state electronics.

The Flash Memory Card is available in 1-Mbyte and 4-Mbyte densities, priced at $298 and $1,198, respectively, in 1,000-piece quantities.

386 SL microprocessor family: Reader Service 38
82360SL I/O subsystem chip: Reader Service 39
Flash Memory Card: Reader Service 40

ADVERTISER INDEX

Alsys, Inc. 29
CACI Products Company 1
Compass '91 58
Data Compression Conference 86
The George Washington University 7
HNC 4-5
IEEE Computer Society Membership 130
IEEE Computer Society Press 27
King Fahd University 39, 89
Local Computer Networks Conference ..... Cover III
Meridian Software Systems, Inc. ..... Cover II
The MITRE Corp. 40
Nanyang Technological Institute 6
Nissan Research & Development, Inc. 125
OOPSLA '91 50
Parallel Processing Conference 111
Schlumberger Well Services 125
Software Engineering Institute 10
StructSoft, Inc. 11
Sun Microsystems 8-9
TeleSoft 12
University of Maryland 127
University of St. Thomas 101
URSIC Computing 119
US WEST Advanced Technologies 49
USENIX Technical Conference ..... Cover IV
Varian Microwave Equipment Products 89
VLDB '91 90
Classified Advertising 120-129

FOR DISPLAY ADVERTISING INFORMATION, CONTACT:

Northern California and Pacific Northwest: John D. Vance & Associates, Inc., 4030 Moorpark Ave., Suite 116, San Jose, CA 95117; (408) 741-0354.

Southern California and Mountain States: Richard C. Faust Co., 24050 Madison St., Suite 101, Torrance, CA 90505; (213) 373-9604.

Midwest: The Kingwill Company, 4433 W. Touhy Ave., #540, Lincolnwood, IL 60646; (708) 675-5755.

East Coast: Atlantic Representative Group, 349 Maple Pl., Keyport, NJ 07735; (908) 739-1444.

New England: Arpin Associates, P.O. Box 6444, Holliston, MA 01746; (508) 429-8907.

Europe: Heinz J. Görgens, Parkstrasse 8a, D-4054 Nettetal 1 - Hinsbeck (F.R.G.); phone: (0 21 53) 8 99 88; fax: (0 21 53) 8 99 89.

Southwest/Southeast: Heidi Rex, 10662 Los Vaqueros Cir., PO Box 3014, Los Alamitos, CA 90720-1264; (714) 821-8380.

For production information, conference, and classified advertising, contact Heidi Rex or Marian Tibayan.

COMPUTER, 10662 Los Vaqueros Cir., PO Box 3014, Los Alamitos, CA 90720-1264; phone (714) 821-8380; fax (714) 821-4010.

PRODUCT INDEX

	RS#	PG#
Advanced Micro Devices	120	107
Alsys, Inc.	7	29
Analog Devices	121	107
Apple Computers	31-36	102
Borland	49	106
C-Cube Microsystems	135	108
CACI Products Company	—	1
Catalyst Semiconductor	122	107
Chips and Technologies	123	107
CMS Enhancements	136-137	108
Compaq	48	106
Cray Research	30	102
Emulex	124, 138	107, 108
Focus Information	139	108
Franklin Datacom	140	108
HNC	2	4-5
Hitachi America	125	107
Int'l CMOS Technology	126	107
Intel	38-40	103
Logic Modeling Systems	127	107
LSI Logic	128-129	107
Meridian Software Systems, Inc.	1	C.II
Microsoft Corp.	25	100
Motorola	130	107
Mountain Network	141	108
Mylex	142	108
NCR	43-44	105
Next Computer	47	106
Object Design	45	106
Ontologic	46	106
Parc Place Systems	21	96
Penny & Giles Controls	26	101
Proteon	143	108
Seiko Instruments USA	23-24	97
Software Engineering Institute	4	10
Software Productivity Solutions	22	96
Stardent Computer	41-42	105
StructSoft, Inc.	5	11
Sun Microsystems	3	8-9
TeleSoft	6	12
URSIC Computing	8	119
US Sage	144	108
Vitesse Semiconductor	131	107
Western Digital	145	108
Xante and Phoenix	146	108
Xerox	37	103

COMPUTER
Reader Service Card

November 1990 issue (Card void after May 1991)
Print address below.

Name _____

Title _____

Company _____

Address _____

City _____

State/ZIP _____

Country _____ Phone _____

Please send (Circle those you want)

201	Publications catalog	**203**	Student membership application
202	Membership information	**204**	Application for senior membership

Reader interest
(Circle what you liked; add comments on the back)

A	Hypercube Systems
B	Correctness Model
C	Reactive Keyboard
D	Knowledge/Databases
E	Application Portability
F	Garnet
G	President's Message
H	Letters to the Editor
J	Standards
K	CS News
L	Update
M	Product Reviews
N	New Products
P	ICs/Microsystems
Q	Conferences
R	Call/Calendar
S	Book Reviews
T	CS Magazines

Product information
(Circle numbers for products and advertisements on which you want more information)

1	21	41	61	81	101	121	141	161	181
2	22	42	62	82	102	122	142	162	182
3	23	43	63	83	103	123	143	163	183
4	24	44	64	84	104	124	144	164	184
5	25	45	65	85	105	125	145	165	185
6	26	46	66	86	106	126	146	166	186
7	27	47	67	87	107	127	147	167	187
8	28	48	68	88	108	128	148	168	188
9	29	49	69	89	109	129	149	169	189
10	30	50	70	90	110	130	150	170	190
11	31	51	71	91	111	131	151	171	191
12	32	52	72	92	112	132	152	172	192
13	33	53	73	93	113	133	153	173	193
14	34	54	74	94	114	134	154	174	194
15	35	55	75	95	115	135	155	175	195
16	36	56	76	96	116	136	156	176	196
17	37	57	77	97	117	137	157	177	197
18	38	58	78	98	118	138	158	178	198
19	39	59	79	99	119	139	159	179	199
20	40	60	80	100	120	140	160	180	200

COMPUTER
Reader Service Card

November 1990 issue (Card void after May 1991)
Print address below.

Name _____

Title _____

Company _____

Address _____

City _____

State/ZIP _____

Country _____ Phone _____

Please send (Circle those you want)

201 Publications catalog
202 Membership information
203 Student membership application
204 Application for senior membership

Reader interest
(Circle what you liked; add comments on the back)

A Hypercube Systems
B Correctness Model
C Reactive Keyboard
D Knowledge/Databases
E Application Portability
F Garnet
G President's Message
H Letters to the Editor
J Standards
K CS News
L Update
M Product Reviews
N New Products
P ICs/Microsystems
Q Conferences
R Call/Calendar
S Book Reviews
T CS Magazines

Product information
(Circle numbers for products and advertisements on which you want more information)

1 21 41 61 81 101 121 141 161 181
2 22 42 62 82 102 122 142 162 182
3 23 43 63 83 103 123 143 163 183
4 24 44 64 84 104 124 144 164 184
5 25 45 65 85 105 125 145 165 185
6 26 46 66 86 106 126 146 166 186
7 27 47 67 87 107 127 147 167 187
8 28 48 68 88 108 128 148 168 188
9 29 49 69 89 109 129 149 169 189
10 30 50 70 90 110 130 150 170 190
11 31 51 71 91 111 131 151 171 191
12 32 52 72 92 112 132 152 172 192
13 33 53 73 93 113 133 153 173 193
14 34 54 74 94 114 134 154 174 194
15 35 55 75 95 115 135 155 175 195
16 36 56 76 96 116 136 156 176 196
17 37 57 77 97 117 137 157 177 197
18 38 58 78 98 118 138 158 178 198
19 39 59 79 99 119 139 159 179 199
20 40 60 80 100 120 140 160 180 200

COMPUTER
Reader Service Card

November 1990 issue (Card void after May 1991)
Print address below.

Name _____

Title _____

Company _____

Address _____

City _____

State/ZIP _____

Country _____ Phone _____

Please send (Circle those you want)

201 Publications catalog
202 Membership information
203 Student membership application
204 Application for senior membership

Reader interest
(Circle what you liked; add comments on the back)

A Hypercube Systems
B Correctness Model
C Reactive Keyboard
D Knowledge/Databases
E Application Portability
F Garnet
G President's Message
H Letters to the Editor
J Standards
K CS News
L Update
M Product Reviews
N New Products
P ICs/Microsystems
Q Conferences
R Call/Calendar
S Book Reviews
T CS Magazines

Product information
(Circle numbers for products and advertisements on which you want more information)

1 21 41 61 81 101 121 141 161 181
2 22 42 62 82 102 122 142 162 182
3 23 43 63 83 103 123 143 163 183
4 24 44 64 84 104 124 144 164 184
5 25 45 65 85 105 125 145 165 185
6 26 46 66 86 106 126 146 166 186
7 27 47 67 87 107 127 147 167 187
8 28 48 68 88 108 128 148 168 188
9 29 49 69 89 109 129 149 169 189
10 30 50 70 90 110 130 150 170 190
11 31 51 71 91 111 131 151 171 191
12 32 52 72 92 112 132 152 172 192
13 33 53 73 93 113 133 153 173 193
14 34 54 74 94 114 134 154 174 194
15 35 55 75 95 115 135 155 175 195
16 36 56 76 96 116 136 156 176 196
17 37 57 77 97 117 137 157 177 197
18 38 58 78 98 118 138 158 178 198
19 39 59 79 99 119 139 159 179 199
20 40 60 80 100 120 140 160 180 200

Editorial comments

I liked: _____

_____

_____

_____

I disliked: _____

_____

_____

I would like to see: _____

_____

_____

_____

For reader-service inquiries, see other side.

PLACE POSTAGE HERE

PO Box is for reader-service cards only.

COMPUTER
Reader Service Inquiries
PO Box 16508
North Hollywood, CA 91615-6508
USA

Editorial comments

I liked: _____

_____

_____

_____

I disliked: _____

_____

_____

I would like to see: _____

_____

_____

_____

For reader-service inquiries, see other side.

PLACE POSTAGE HERE

PO Box is for reader-service cards only.

COMPUTER
Reader Service Inquiries
PO Box 16508
North Hollywood, CA 91615-6508
USA

Editorial comments

I liked: _____

_____

_____

_____

I disliked: _____

_____

_____

I would like to see: _____

_____

_____

_____

For reader-service inquiries, see other side.

PLACE POSTAGE HERE

PO Box is for reader-service cards only.

COMPUTER
Reader Service Inquiries
PO Box 16508
North Hollywood, CA 91615-6508
USA

Two new series expand Stardent's offerings

The 500 and 3000VS series from Stardent Computer include four low-end desktop visualization systems and six new high-end systems.

Stardent says its 500 "Stiletto" Series is the lowest price dual-processor series of supercomputers and visualization systems available. Prices start at less than $20,000. The 500 Series includes desktop visualization and computational supercomputers and departmental servers, all available in one- or two-processor models.

All products in the 500 series contain a MIPS R3000 32-MHz integer chip, a MIPS R3010 scalar floating-point unit, and an Intel i860 for emulating vector instructions. The graphics subsystem uses two i860 processors for 3D color pixel and polygon processing. The system delivers 32 peak MIPS and 48 peak Mflops per processor, for a dual-processor total of 64 peak MIPS and 96 peak Mflops, according to the company. Graphics-rendering performance is up to 40,000 100-pixel gouraud-shaded triangles per second, Stardent says.

The operating system is based Unix System V.3, and the computer supports all applications for the Stardent 3000 Series.

The 3000VS Series is based on the company's VX graphics subsystem. It lets users construct realistic scenes using high-resolution texture mapping, real-time compositing, and multipass "true" transparency, according to the company. The VX subsystem comprises 24 custom ASICs that provide hardware implementations of graphics features normally executed by software. The subsystem also offers full, double-buffered, 24-bit color and proprietary global antialiasing to let users antialias entire scenes.

The series supports one to four computational units composed of a MIPS-based R3000 32-MHz RISC microprocessor for integer processing, a MIPS R3010 scalar floating-point coprocessor, and a custom vector-processing unit, delivering peak performance of 128 Mflops in a four-processor configuration, according to Stardent. It offers up to half a gigabyte of main memory to support memory-intensive applications.

The 3000VS is available in single- or dual-seat configurations. In the latter, each user has a complete system, including a graphics processor, CRT, keyboard, and mouse. Users can work in parallel on the same 3000VS system and have instantaneous access to shared data and computational results.

Both series are source compatible with Stardent's existing products and binary compatible with the 3000 Series.

Prices for the Stardent 500 range from $17,900 for the single-processor, diskless, OEM version to $70,000 for a fully configured, dual-processor, end-user version. An "average" $39,000 configuration would include a single-processor system with 16-Mbyte main memory, 250-Mbyte disk storage, 120-Mbyte internal tape storage, 16-inch color monitor, keyboard, and mouse.

One of Stardent's 500 "Stiletto" series.

Prices for the Stardent 3000VS range from $100,000 to $300,000. Current 3000 customers can get a field upgrade to the 3000VS for $30,000.

500 Series: Reader Service 41
3000VS Series: Reader Service 42

NCR announces System 3000 computers, "cooperative" software

NCR's new general-purpose computers span a power range of from 7.5 MIPS to more than 100,000 MIPS and support NCR Unix V.4, SCO Unix, OS/2 and MS-DOS.

Overall, the System 3000 includes three uniprocessor levels and four multiprocessor levels. The six systems available now include uniprocessor desktop workstations and servers, and uniprocessor and multiprocessor servers. These systems represent a power range of 7.5-320 MIPS. Prices for the available systems range from $10,000 to $800,000.

NCR also announced its Cooperation software environment, the first software product under NCR's Open, Cooperative Computing Architecture.

Cooperation's first release runs on 286-, 386-, or 486-based PCs, either ISA or MCA bus, running DOS at the client workstation, and MCA computers running current releases of OS/2 SE and NCR Unix V.4 at the server. NCR intends to support OS/2 and Unix client workstations in later releases.

Cooperation comprises 54 software modules in four major categories: information access services, user services, application services, and network services. According to NCR, Cooperation allows access to applications, files, and databases across multivendor systems via such features as remote application access (using terminal emulators), a communications authoring tool, and database access (using SQL database query tools and gateways).

Cooperation features an object-oriented graphical user interface, task automation, a compound document editor, an information storage manager, a business information monitor, and workflow automation.

The system also includes a mail facility that can exchange messages through either an X.400 mail gateway or a Softswitch gateway that supports mail interchange among most e-mail systems. Within a "cooperative" environment, the mail facility lets users send and receive such objects as text, data, images, or file folders.

Cooperation supports token-ring and Ethernet local area networks using LAN Manager. Wide area network gateways are provided for OSI, SNA, and X.25 networks; NCR Net Manager manages these.

Cooperation modules can be purchased separately or in preconfigured packages. Pricing ranges from $23,000 to $37,000 for a 12-user configuration. Pricing for a 24-user configuration ranges from $40,000 to $58,000. General availability for OS/2 systems with DOS clients is scheduled for March 1991. Availability for Unix V.4 servers with DOS clients is scheduled for August 1991.

System 3000: Reader Service 43
Cooperation: Reader Service 44

Database management system features object orientation

The Object Store database management system by Object Design is based on C++ and includes a database engine, professional development tools, and graphical end-user interface tools.

The software is designed for applications in mechanical and electrical CAD, computer-aided publishing, computer-aided software engineering (CASE), mapping and geographic information systems, financial and scientific modeling, and advanced office automation.

Object Store applications are compatible with such subroutine library languages as Fortran, Pascal, and assembly language, according to Object Design.

Object Store also features C and C++ development tools for building large-scale applications, including a high-level programming interface, a browser, a source-level debugger, and a "Schema Designer," which is an interactive graphical tool for developing database schemas and generating the associated C++ code.

The software runs on Sun 3 and Sparcstation computers. Object Designs also expects to port Object Store to other Unix workstations and Microsoft Windows 3.0 personal computer platforms by the first quarter of 1991.

Object Store is priced "per seat" from $2,000 to $9,000, depending on the number of seats and licensed sites. Runtime licensing is also available.

Reader Service 45

68040 systems next for Next

Next Computer has introduced four products based on Motorola's 68040 microprocessor.

The company says its Nextstation is more powerful and compact than the original Next computer and costs less than $5,000. The other systems are the Nextstation Color, a 16-bit, Postscript color version of the Nextstation; the Nextcube, a configurable, expandable Next computer that can be used as a network server or high-end desktop computer; and the Nextdimension, a 32-bit Postscript color system.

All the systems include Release 2.0 of the Nextstep graphical user interface and development environment, a Unix-based operating system, Next's multimedia e-mail system, digital signal processing chip, Display Postscript, an internal 2.88-Mbyte floppy disk drive and 105-Mbyte hard disk drive, built-in Ethernet and TCP/IP networking, and Megapixel Display (color or monochrome).

Release 2.0 of the system software includes new support for color, DOS floppy disks, international versions, the 68040, and twisted-pair Ethernet. Next says applications written for Release 1.0 run faster but unmodified on Release 2.0.

The Nextstation's base configuration is $4,995, and the Nextcube system is $7,995. Both are available this month. The Nextstation Color system and Nextdimension board are $7,995 and $3,995, respectively, and are scheduled for the first quarter of 1991.

Reader Service 47

Ontos Release 2.0 provides distributed object system

Ontos Release 2.0 is an upgrade of Ontologic's object-oriented database management system for interactive, network-based, data-intensive applications.

The new version fully supports multiple inheritance and C++ 2.0 as well as full data distribution across local area networks, according to Ontologic. The company also says the new release offers better specialized transaction support for design applications.

Ontos 2.0 includes a graphical design tool that automatically generates C++ header files and database schema, and it features a graphical browser for interactive database inspection and maintenance. In addition, the new release now offers customized storage for small objects and continues to offer storage for medium and large objects.

Ontos 2.0 is available on several platforms, including Sun 3, Sun 4/Sparc, Apollo, DECstation, and OS/2.

Reader Service 46

Notebook PC based on new technology

Compaq says its LTE 386s/20 notebook computer is the first commercial computer product to use "rigid flex" technology that incorporates both rigid and flexible materials in one continuous sheet. The computer's circuit board thus folds in half to save space and uses flexible interconnections instead of detachable cables.

The 8.5¥11-inch, 7.5-pound notebook PC is based on a 20-MHz Intel 386SX. The computer includes either a 30- or 60-Mbyte fixed disk drive, VGA display, a cache memory controller, a 3.5-inch floppy disk drive, 2 Mbytes of RAM (expandable to 10 Mbytes), a nickel-cadmium battery pack, and an AC adapter.

The Compaq LTE 386s/20 with a 60-Mbyte hard drive costs $6,999. With a 30-Mbyte drive, the machine is $6,499.

Compaq has also introduced an optional Desktop Expansion Base with two additional expansion slots. Users can slide the new notebook unit into the base to gain access to external storage devices, VGA monitors, or keyboards. The base unit costs $1,499.

Other new options for the machine include a 2400-baud modem, 1- and 4-Mbyte memory cards, a CD-ROM adapter, a serial interface board, and external battery charger, and a car adapter.

Reader Service 48

Quattro Pro gets upgrade

Borland's Quattro Pro 2.0 spreadsheet program offers such new features as "Solve For," a goal-seeking technology for solving "what-if" scenarios; database connectivity to let users manipulate data in Borland's Paradox database program and in mainframe computers; and enhanced presentation graphics, including the ability to create "branching" slide shows and 3D graphs.

Quattro Pro 2.0 costs $495, and the LAN pack for network use costs $295. Registered owners of version 1.0 can upgrade through Borland for $79.95.

Reader Service 49

IC Announcements

Company, Model, Function	Comments	R.S. No.
Advanced Micro Devices Am286ZXTM/LXTM Motherboards on a chip	Integrated microprocessors incorporate all components of the original IBM AT motherboard on one CMOS IC. Both processors will be offered initially at speeds of 12 and 16 MHz. Sampling will begin in fourth quarter 1990, with production scheduled for second quarter 1991. Cost (1,000s): from $69 for the Am286ZX-12 to $89 for the Am286LX-16.	120
Analog Devices AD1864 Audio DAC	Dual 18-bit DAC is intended for high-performance, multichannel digital audio applications such as electronic musical instruments and multimedia workstations. It guarantees minimum 110-dB channel separation and maximum 0.0025% total harmonic and noise distortion. It dissipates less than 225 mW and interfaces to popular digital filters. Cost (1,000s): $14.25.	121
Catalyst Semiconductor CAT28C16V3 EEPROM	CMOS 16-Kbyte (2-Kbyte × 8) low-power, 3V parallel EEPROM is designed for high-volume portable applications, such as laptop PCs and cellular telephones, that require a simple interface for in-system programming. Features 10 mA active and 50 mA standby maximum current ratings, on-chip address and data latches, and self-timed write cycle with auto-erase. No cost given.	122
Chips and Technologies ELEATsx Desktop chip set	Three-chip set consists of the company's SCATsx systems logic controller, the 82C450 1-Mbyte DRAM VGA controller, and the 82C711 universal peripheral controller, which integrates two serial ports and one parallel port with a floppy disk controller and hard disk drive interface on a single chip. Now sampling; production quantities available first quarter 1991. Cost (1,000s): $68.	123
Emulex FAS200 Microchip controller	The first in the company's series of microchip controllers supporting the SCSI-2 standard, the FAS200 offers sustained synchronous data-transfer rates between the computer and the peripheral device of up to 10 Mbytes/sec. Cost (1,000s): $22.30 or $27.50, depending on the number of pins on each microchip. Available in sample quantities.	124
Hitachi America HN58C256, etc. ROM, EPROMs, EEPROMs	Family of nonvolatile memory devices includes 256-Kbyte and 1-Mbyte EEPROMs, 1-Mbyte and 4-Mbyte UV EPROMs, and a 4-Mbyte mask ROM. The devices address diverse applications ranging from POS terminals to automatic telephone dialers; their 0.8-µm fabrication process permits high density with small die sizes. Cost (1,000s): $7-$68.	125
Int'l CMOS Technology 93C46A EEPROM	One-Kbyte (64 × 16) CMOS serial EEPROM is targeted for low-cost, low-power microcomputer-based systems such as VCRs, pagers, meters, and automobile systems. Features 2-MHz maximum operating frequency and eight-pin DIP or surface-mount packaging. Cost (1,000s): $0.70.	126
Logic Modeling Systems 68040 Logic Model	Full-function simulation model of the Motorola 68040 microprocessor is the latest addition to Logic Modeling's library of 32-bit microprocessors, which already includes the other members of Motorola's 68000 family. Allows system and ASIC designers to model behavior and execute actual system code during design and system verification. No cost given.	127
LSI Logic Ngine RPM3310 RISC subsystem	Module includes LSI Logic's LR3000 MIPS RISC microprocessor, LR3010 MIPS floating-point accelerator, and LR3220 read-write buffer. It resolves — on board — the complex timing constraints of RISC-based architectures. Prototypes are now available with production quantities scheduled for end-of-the-year shipment. Cost (5,000s): under $1,000.	128
LSI Logic L64700 series Video compression chips	Seven chips encompass all functions necessary for compression and decompression of full-motion video in real time. Chips adhere to the Px64 (CCITT H.261) video-teleconferencing and video telephone standard and the standard being developed by the Motion Picture Experts Group. Five chips are available now; two will be later this month. Cost (1,000s), kit: $700.	129
Motorola 68331 Microcontroller	Newest member of the 68300 family for embedded control is similar to the 68332, but it is designed for applications with less emphasis on real-time control and features cost-saving changes. The 331 integrates a subset of the 68020 (CPU32) with a queued serial module, a system integration module, and a general-purpose timer. Now sampling. Cost (1,000s): $36.	130
Vitesse Semiconductor Megaswitch VSC864 GaAs crosspoint switch	VLSI gallium-arsenide 64 × 64 crosspoint switch, supporting data rates up to 200 Mbytes/sec., is intended for high-speed digital applications including data distribution for telecommunications, computer networks, and multiprocessing environments. Operates from a −2V power supply, dissipates less than 8W power, features industry-standard ECL I/O. Cost (100s): $1,196.	131

Microsystem Announcements

Company, Model, Function	Comments	R.S. No.
C-Cube Microsystems Compression Master Add-in boards	The Compression Master and Compression Master PC, using JPEG, are add-in boards for Apple Macintosh, IBM PC AT, and compatible computers. Both boards compress images at ratios up to 24:1 with no visible loss of quality at speeds that do not interfere with routine computing activities. Cost: $995 each.	135
CMS Enhancements Enhance RAM Memory adapter	A memory adapter for the PC AT and compatibles and the IBM PS/2 Model 30 286. Provides up to 8 Mbytes of expanded, enhanced, and backfill memory. Supports systems with speeds up to 33 MHz and uses a 16-bit AT bus interface. Requires no hardware changes. Cost: $299.	136
CMS Enhancements Memory modules	Modules that support a wide variety of systems, including IBM PS/2, Compaq, Macintosh, AST, and Toshiba. Includes modules with capacities of up to 4 Mbytes of memory. No price given.	137
Emulex PC01 Smart Unix Serial Card Front-end communications processor	The serial card enables IBM or compatible PC workstations to be used as platforms for multiuser Unix or Xenix systems. Up to 16 users can be connected with all lines operating at 38.4 Kbps. Up to three PC01s can be installed in a system. The DCP-286/MC communications board OEM applications include high-speed X.25 and SNA links. Cost: $1,195 to $1,595, PC01 (VARs); $1,895, DCP-286/MC (OEMs).	138
Focus Information Systems 2 the Max VGA 4000 Super VGA card	Flexible memory expandable to 1 Mbyte and 1,024 × 768-pixel resolution with up to 256 displayable colors in all standard and extended graphic modes for IBM-compatible PCs. The card has dot frequency of up to 65 MHz in graphic mode and 56 MHz in text mode, with a 70-Hz screen-refresh rate. Cost: $295 to $395.	139
Franklin Datacom Model MPP-U Front-end processor	For IBM RS/6000 and PS/2 computers running AIX, Unix, or DOS applications. Up to eight boards with four ports each can be installed in a Micro Channel architecture, permitting up to 32 ports at up to 64 Kbps on each port, or 2.048 Mbps throughout. No cost given.	140
Mountain Network Solutions File Safe 8500 Tape drive	A high-capacity minicartridge tape subsystem with the Integrated Device Electronics (IDE) embedded AT interface. Storage capacity is up to 304 Mbytes in a 3.5-inch half-height form. Throughput rate is up to 10 Mbytes per minute in data compression mode. Evaluation units in January 1991, followed by production quantities in March. Cost: $895.	141
Mylex MAE486-33 33-MHz system board	The EISA Bus system board is designed for high-end applications in a multiuser, multitasking environment. The product has a 128-Kbyte write-back cache in addition to the 8-Kbyte cache on the 486 processor chip. It also includes a socket for the Weitek 4167 floating-point coprocessor. Cost: $4,620 to $5,200, depending on volume.	142
Proteon p1890 Pronet-4/16 Micro Channel adapter	For file-server and high-performance networking applications. Lifetime warranty for all Pronet-4/16 Mbps network interface cards. Supports up to 72 nodes per ring in 4/16 Mbps UTP environments. Cost: $850.	143
US Sage Main Lan for PS/1 Ethernet adapter card	The 10 Mbps Ethernet adapter card networks IBM PS/1 computers and enables linkage in either peer-to-peer or file server environments. Cards are compatible with Main Lan's own operating system and with Novell. PS/1 needs only 30 Kbytes for conventional PC workstations, or 15 Kbytes in workstations with EMS, and runs with Microsoft Windows. Cost: $499 (starter pack); $599 (2-4 users); $999 (2-8 users); and $1,499 (unlimited use).	144
Western Digital WDAB130 and WDAH260 Intelligent disk drives	AT-compatible 2.5-inch drives. The WDAB130 has a formatted capacity of 31.5 Mbytes with a 19-ms average seek time, is 0.6 inches high, and weighs 144 grams. The WDAH260 has a formatted capacity of 62.9 Mbytes with a 19-ms average seek time, is 0.75 inches high, and weighs 160 grams. Volume production first quarter 1991. Cost: $325, WDAB130; $495, WDAH260 (evaluation units).	145
Xante and Phoenix Technologies Power Script controller	Printer-resident controller uses Phoenix Page Postscript-compatible interpreter. Adds up to 10 times more Postscript processing performance to laser writers, full compatibility with Adobe Type 1 fonts, and resolution enhancements of up to 300 × 1,200 dpi. Cost: $1,695.	146

CONFERENCES

Editor: Edmund L. Gallizzi, Computer Science Dept., Eckerd College, St. Petersburg, FL 33733, phone (813) 864-8272, Compmail e gallizzi

Parallelism must be exploited at several levels, ICPP keynoter asserts

David J. Lilja, University of Illinois at Urbana-Champaign

Most existing parallel computer architectures support only homogeneous, fine-grain parallelism, while most problems are actually parallel at several levels, H.T. Kung pointed out when he keynoted the 19th International Conference on Parallel Processing. The conference was held August 13-17 in St. Charles, Illinois.

ICPP 90 featured three days of presentations in four simultaneous tracks (two in architecture, one in software, and another in algorithms and applications) and two days of tutorials. It was sponsored by Pennsylvania State University. Benjamin W. Wah, David A. Padua, and Pen-Chung Yew from the University of Illinois served as conference cochairs. Tse-yun Feng of Penn State provided overall coordination.

In his keynote address, Kung of Carnegie Mellon University asserted that both speech and vision systems require low-level signal processing combined with some high-level processing, such as natural language parsing or object identification. In addition, exploiting only one level of parallelism limits the scalability of current parallel architectures.

Kung described the Nectar (Network Computer Architecture) project at Carnegie Mellon, which implements a flexible, extensible network backplane to integrate heterogeneous machines into one parallel system. This system can exploit parallelism on two levels simultaneously: fine-grain parallelism for tasks on individual processing nodes and coarse-grain parallelism spread among separate tasks executing on different nodes.

Nectar interconnects heterogeneous systems via dedicated communication processors at each node. Fiber optic technology provides low-latency, high-bandwidth communication between nodes and allows cost-effective extension of the system. A system prototype is in operation, and significant efforts are being directed at performance analysis and software development.

Recognizing the popularity of the moderated panel sessions, the conference expanded the number to five. In the session on "High-Performance Computing," Ron Bailey of NASA pointed out the need for teraflop performance, exaword (10^{18} words) storage, and greatly improved input/output capacity to handle envisioned problems at NASA. The panelists' hope was that parallel processing technology will be able to provide the necessary performance.

Echoing Kung's keynote remarks, the answer to the question posed by the panel "Can a Large-Scale Parallel System Be a Commercial Success?" seemed to be "not until parallel systems are easier to use, have more application software available, and can be easily integrated into production environments."

In the "Parallel Processing for Nonnumeric Applications" panel session, the consensus was that nonnumeric problems are much harder to parallelize than numeric problems. One panelist suggested that dynamic scheduling is much more important to nonnumeric problems than to numeric problems due to the former's irregular control and data flow.

The remaining panel sessions were entitled "Applications of Parallel Processing: The Next 20 Years" and "Contrasting Conceptual Frameworks for Parallel Programming Environments."

DAC panel explores plight of women in microelectronics

Petra Michel and Ulrich Lauther, Siemens AG

The status of women in microelectronics is unacceptable. This fact came into focus at a Women in Microelectronics panel discussion June 25 during the 27th Design Automation Conference. Sponsored by the IEEE Computer Society Design Automation Technical Committee and the ACM Special Interest Group on Design Automation, DAC 90 was held June 24-29 in Orlando, Florida.

To achieve a balance between industry and academia, between men and women, and between participants from the US, Asia, and Europe, the panel's organizer, Ulrich Lauther of Siemens AG, Munich, Germany, invited the following to serve on the panel: Petra Michel, also of Siemens AG, session chair; Aart J. de Geus, Synopsys; Satoshi Goto, NEC; Kuniko Kubono, Fujitsu Laboratories; Alice Parker, the University of Southern California; and Gabriele Saucier, Institute National Polytechnique de Grenoble.

Michel presented data from the United Nations office in Geneva on the plight of women in general and in the electronics industry in particular. She said that, although women constitute slightly more than 50 percent of the worldwide population, they represent only a third of the official labor force, contribute two-thirds of all working hours, earn a tenth of the world's income, and own one percent of all property. Two-thirds of the world's illiterates are female, Michel added.

In legislative bodies worldwide, 9.7 percent of the members are women, and about 4 percent of the decision-makers at ministerial levels are female, Michel said.

In industry, the situation seems to be more positive — at least at first glance. At IBM, for instance, women constitute about 20 percent of all employees and about 13 percent of management, said Michel. However, these numbers do not shed any light on women in management in general. Figure 1, for example, provides more detailed data for Philips and Siemens AG in Europe. Although 27.9 percent of the employees at Philips and 16 percent at Siemens AG are female, the fractions drop, respectively, to 5.6 percent and 5.7 percent among professionals and to 0.6 percent and 0.7 percent in senior management. In both companies, top management is exclusively male.

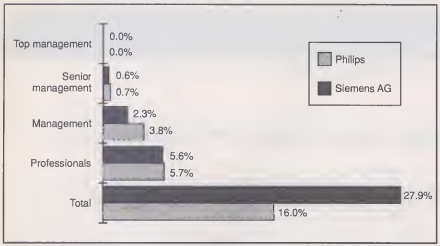


Figure 1. Female participation in management.

Nevertheless, the shortage of personnel in microelectronics requires that women become increasingly involved. In Japan, for example, the number of female university graduates has continually increased from 2,000 in 1955 to more than 25,000 in 1989, according to data Goto cited. He said there are now more female graduates than male graduates in Japan. Some 23 percent of all NEC employees are women, but only 0.2 percent of managers are women, Goto added.

Due to a general shortage in the work force, more than 50 percent of new employees at Fujitsu are women, according to Kubono, who added that only 16 of the 4,000 managers are women.

Such discrepancies can be attributed only partly to the different age distributions for men and women, as shown for Siemens in Figure 2. At NEC, for instance, 50 percent of female employees leave their companies within the first five years of employment, and another 25 percent leave within the first eight years.

The panelists offered varied explanations of why women leave their companies early, including objections by husbands, unacceptable societal expectations, a lack of child-care programs, and the large amounts of overtime expected from employees in electronics companies. This does not suggest that women cannot work overtime, but that they cannot, as society seems to expect, take prime responsibility for child care and the elderly and simultaneously put in many overtime hours.

Parker said that, although men often overemphasize pregnancy as a problem, the real quandary is over responsibility for rearing children, a commitment that can last more than 20 years. This responsibility should be shared equally by female and male employees, but neither academia nor industry has been sensitive to the needs of working parents. In contrast to most European countries, US employers don't offer leaves to care for sick children, maternity leaves are inadequate, and paternity leaves are not possible, Parker said. In all countries, work hours are too inflexible. Another problem discussed less frequently is care for the elderly, a responsibility that society places primarily on female family members, said Parker.

The first step toward solving these problems must be to foster a greater awareness of these issues among both men and women. At least in the US, de Geus reported, small startup companies seem more inclined than larger firms to provide solutions, including gender-neutral parental-leave models, flextime, or day-care programs.

The panel's general discussion mainly concerned such education-linked questions as how women can be encouraged to go into engineering and whether male and female students should be treated differently (for example, eliminate co-education). Although there is some evidence that gender-specific education might be effective (for example, a very high percentage of women engineers come from women-only colleges), the panel's consensus was that, for various reasons, women oppose any gender-specific advantages in professional life.

On the other hand, Michel said women do maintain one advantage due to their small representation in the work force, especially in higher positions. She said that, because it is unusual to see female professionals in the microelectronics industry at all, they are observed more closely and are, therefore, recognized more quickly if they qualify for higher positions. Women should not hesitate to take advantage of this, Michel said.

Unfortunately, the panelists could not answer some of the questions raised, nor provide complete solutions. Other issues (for example, the question of whether women in management act differently from men) were either not addressed or only touched on briefly.

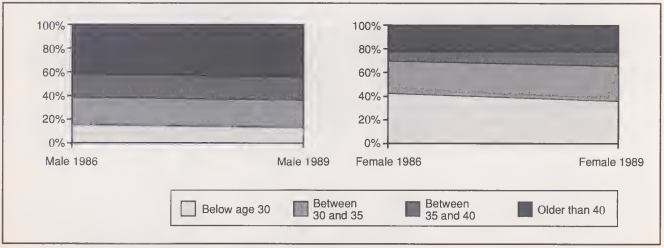


Figure 2. Age distribution of male and female professionals at Siemens AG.

CALL FOR PAPERS

— 20th Annual Conference —

1991 International Conference on Parallel Processing

August 12–16, 1991

The Pennsylvania State University

INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING

Authors are invited to submit papers describing recent advances on all aspects of parallel/distributed processing. These may include parallel/distributed logic circuits, impact of VLSI to parallel processor architecture; various concurrent-, parallel-, pipeline-, or multiple-processor architectures; processor-memory interconnections; computer networks; distributed databases; reliability and fault tolerance; modeling and simulation techniques; performance measurements; operating systems; languages; compilers; algorithms; or various application studies.

INSTRUCTIONS FOR AUTHORS
The deadline for submitting papers is January 10, 1991.

Submit five (5) copies each of a 100-word abstract and the full text (including five specific key words) to one of the following Technical Program Co-Chairpersons according to their areas of responsibility:

ALGORITHMS AND APPLICATIONS papers to:
Kimming So (512) 823-2913 (Off.)
IBM Austin (512) 823-3062 (Sec.)
Internal Zip 2812 (512) 823-2948 (Fax)
11400 Burnet Road sok@ibm.com
Austin, TX 78758

SOFTWARE-ORIENTED papers to:
Herbert D. Schwetman (512) 338-3428 (Off.)
MCC (512) 338-3600 (Fax)
3500 West Balcones Center Drive hds@mcc.com
Austin, TX 78759

HARDWARE-ORIENTED AND OTHER papers to:
C. L. Wu (512) 471-4085 (Off.)
The University of Texas (512) 471-5532 (Fax)
at Austin clwu@emx.utexas.edu
Dept. of Electrical and Computer Eng.
Austin, TX 78712

Please limit the length of your paper to about 20 double-spaced typing pages, and include your office and/or home phone numbers as well as your electronic mail address (if available). Authors who wish to keep their names from reviewers should make their desires explicit when submitting their papers. Submitted papers will be acknowledged promptly.

Selected papers are published as regular, short or poster papers. Authors will be notified of acceptance by March 20, 1991. Due to the large number of papers we receive every year, authors are expected to help in reviewing papers for the conference.

Please note the deadline for submitting papers is **January 10, 1991.**

CONFERENCE PROCEEDINGS
The regular papers, the summaries of the short papers, and the abstract of the poster papers will be published in the conference proceedings. Special sheets for the preparation of accepted papers for the proceedings will be sent to each author.

CONFERENCE AWARDS
The conference will give two awards: one for the **Daniel L. Slotnick Award for Most Original Paper,** and the other for **Best Presentation**. In addition, a small number of awards may be given for Outstanding Papers and Distinguished Presentations.

CONFERENCE ENVIRONMENT
The Conference is tentatively scheduled at the Pheasant Run Resort in St. Charles, Illinois. Pheasant Run is only 45 minutes from Chicago's loop and 25 miles from O'Hare International Airport.

The Resort is located in **DuPage County, Illinois**, and is situated on 300 acres in the beautiful Fox River Valley near St. Charles, Illinois. The 500 newly renovated rooms are homey and gracious with scenic views of a small lake or a beautiful golf course. On a clear day the Chicago skyline is visible from the upper levels of its sleek 15 story Tower. Special entertainment is available nightly.

There is transportation available through the Hotel Transportation Department between Pheasant Run and O'Hare International Airport (or Midway Airport).

Informal, open bar gatherings will be held nightly for the conference participants.

A conference brochure with pre-registration form and advance technical program will be prepared and mailed in May/June 1991.

TUTORIALS
Pre-conference (on August 12) and post-conference (on August 16) tutorials relating to parallel processors and processing will be offered.

SPECIAL EVENTS
A number of special events and activities are being planned to celebrate the 20th Annual Conference.

CALL FOR PAPERS

IEEE Software seeks articles for publication in 1991 and 1992. Submit eight copies of article to Carl Chang, *IEEE Software*, 1120 Science and Eng. Offices, MC 154, Univ. of Illinois, Chicago, IL 60680, Internet ckchang@uicbert.eecs.uic.edu.

Sixth IEEE Conf. on Structure in Complexity Theory: June 30-July 3, 1991, Chicago. Submit 10 copies of extended abstract or full paper to N. Immerman, Computer and Information Science Dept., Univ. of Massachusetts, Amherst, MA 01003, e-mail immerman@cs.umass.edu.

Int'l J. of Systems Automation: Research and Applications will begin publication in January 1991. Sponsor: Int'l Soc. for Productivity Enhancement. For submittal instructions, contact Biren Prasad, PO Box 731, Bloomfield Hills, MI 48303-0731, phone (313) 265-5749.

ISCA 18, 18th Int'l Symp. on Computer Architecture: May 27-30, 1991, Toronto. Cosponsor: ACM. Submit five copies of manuscript by **Nov. 21, 1990**, to John Hayes, Electrical Eng. and Computer Science Dept., Univ. of Michigan, 1301 Beal Ave., Ann Arbor, MI 48109, phone (313) 763-0386.

Fourth Int'l Conf. on Networks and Electronic Office Systems: Nov. 12-14, 1991, London. Sponsor: Institution of Electrical Engineers. Submit synopsis by **Nov. 26, 1990,** to Conf. Services, IEE, Savoy Place, London WC2R 0BL, UK, fax 44 (71) 240-7735.

Int'l Conf. on Symbolic-Numeric Data Analysis and Learning: Sept. 17-20, 1991, Paris. Sponsor: Institut National de Recherche en Informatique et en Automatique. Submit paper by **Nov. 30, 1990,** to Conf. Secretariat, INRIA, Service des Relations Exterieures, Domaine de Voluceau — Rocquencourt, BP 105, 78153 Le Chesnay Cedex, France; phone 33 (1)-39-63-55-00, fax 33 (1) 39-63-56-38.

SID 91, 1991 Int'l Symp., Seminar, and Exhibition: May 6-10, 1991, Anaheim, Calif. Sponsor: Soc. for Information Display. Submit abstract and technical summary by **Nov. 30, 1990,** to Jay Morreale, Palisades Inst. for Research Services, 201 Varick St., Suite 1140, New York, NY 10014, phone (212) 620-3371, fax (212) 620-3379.

ECOOP 91, Fifth European Conf. on Object-Oriented Programming: July 15-19, 1991, Geneva, Switzerland. Submit five cop-

Call for papers and referees for *Computer*

Computer seeks articles for inclusion in upcoming special issues.

Distributed Computing Systems has been selected as the theme for the **August 1991** issue. Prospective authors are invited to submit tutorial, survey, descriptive, case-study, application-oriented, or pedagogic manuscripts. See the July 1990 issue of *Computer* (p. 120) for complete information.

The deadline for full manuscripts is **January 1, 1991**. Notification of decisions is set no later than **March 15, 1991**, and the final version of each manuscript is due no later than **May 1, 1991**.

Submittals and questions should be directed to either of the guest editors, Mukesh Singhal, Dept. of Computer and Information Science, Ohio State University, Columbus, OH 43210, phone (614) 292-5839, e-mail singhal@cis.ohio-state.edu; or Thomas L. Casavant, Dept. of Electrical and Computer Eng., University of Iowa, Iowa City, IA 52242, phone (319) 335-5953, e-mail tomc@eng.uiowa.edu.

Multimedia Information Systems will be the theme of the **October 1991** issue. Manuscripts reporting survey, original research, design and development, and applications of multimedia technology are sought. See the October 1990 issue of *Computer* (p. 100) for complete information.

An abstract of each manuscript is due no later than **December 1, 1990**, and eight copies of each full manuscript must be submitted by **February 1, 1991**. Notification of decision is set **May 1, 1991**, and the final version of each manuscript must be submitted by **July 1, 1991**.

Submissions and questions should be directed to A. Desai Narasimhalu, Inst. of Systems Science, National University of Singapore, Heng Mui Keng Terrace, Singapore 0511, phone (65) 772-2002, fax (65) 778-2571, e-mail issad@nusvm; or Stavros Christodoulakis, Department of Computer Science, Davis Building, University of Waterloo, Waterloo, Ont., Canada, phone (519) 888-4452 or (30) 821-64846, fax (519) 885-1208 or (30) 821-53571, e-mail schristodoul@waterloo.edu.

For submittal to Computer, *manuscripts must not have been previously published or currently submitted for publication elsewhere. Each manuscript should be no more than 32 typewritten, double-spaced pages long, including all text, figures, and references. Each submittal should include a cover page that contains the title of the article, the full name(s) and affiliation(s) of the author(s), complete postal and electronic address(es) of all the authors as well as their telephone and fax number(s), a 300-word abstract, and a list of keywords identifying the central issues of the manuscript's contents. The final manuscript should be approximately 8,000 words in length and contain no more than 12 references.*

If you are willing to review articles for any of these special issues, please send a note listing your research interests to Jon Butler, associate technical editor of Computer, *or to one of the guest editors listed for the particular issue. Butler may be reached at the Department of Electrical and Computer Engineering, Naval Postgraduate School, Code EC/Bu, fax (408) 646-2760, e-mail butler@cs.nps.navy.mil.*

Heterogeneous Distributed Database Systems is the theme planned for the **December 1991** issue. See the August 1990 issue of *Computer* (p. 115) for complete information.

Abstracts of the manuscripts are due no later than **January 1, 1991**, and eight copies of the full manuscripts must be submitted by **April 1, 1991**. Notification of decisions is **July 1, 1991**, and the final version of each manuscript is due **September 1, 1991**.

Submissions and questions should be directed to Sudha Ram, Department of Management Information Systems, Eller School of Management, University of Arizona, Tucson, AZ 85721, phone (602) 621-2748, e-mail ram@mis.arizona.edu on Internet or ram@arizmis on Bitnet.

ies of paper by **Nov. 30, 1990,** to Pierre America, Philips Research Labs, PO Box 80.000, 5600 JA Eindhoven, The Netherlands.

Symp. on Solid Modeling Foundations and CAD/CAM Applications: June 5-7, 1991, Austin, Texas. Sponsor: ACM SIGGraph. Submit five copies of paper by **Nov. 30, 1990,** to Jaroslaw Rossignac, J2-C03, IBM T.J. Watson Research Center, PO Box 704, Yorktown Heights, NY 10598, phone (914) 784-7630, fax (914) 784-7455, e-mail jarek@ibm.com.

20th Mumps Users Group Meeting: June 3-7, 1991, New Orleans. Submit eight copies of draft manuscript by **Nov. 30, 1990,** and final manuscript by **Mar. 7, 1991,** to Mumps Users Group, 4321 Hartwick Rd., Suite 100, College Park, MD 20740, phone (301) 779-6555, fax (301) 779-7674.

Int'l J. Computer-Aided VLSI Design plans a special issue on hardware accelerators for computer-aided design. Publisher: Ablex. Submit five copies of full paper by **Dec. 1, 1990,** to Ausif Mahmood, Electrical Eng. Dept., Washington State Univ. at Tri-Cities, 100 Sprout Rd., Richland, WA 99352, phone (509) 375-9234.

1991 ACM Int'l Conf. on Supercomputing: June 17-21, 1991, Cologne, Germany. Co-sponsors: Gesellschaft für Informatik et al. Submit five copies of full manuscript by **Dec. 1, 1990,** to Elias Houstis, Computer Science Dept., Purdue Univ., West Lafayette, IN 47907 (for the Americas); Ulrich Trottenberg, GMD/F1, Schloss Birlinghoven, D-5205 Sankt Augustin 1, Germany (for Europe and Africa); or Toshitugu Yuba, ETL 1-1-4 Umesono, Tukuba, Ibaraki, 305 Japan (for Japan and the Far East).

Computer Animation 91: May 22-25, 1991, Geneva, Switzerland. Cosponsors: Computer Graphics Soc. Submit four copies of paper by **Dec. 1, 1990,** to Daniel Thalmann, Computer Graphics Lab, Swiss Federal Inst. of Technology, CH 1015, Lausanne, Switzerland, phone 41 (21) 693-5214, fax 41 (21) 693-3909, e-mail thalmann@eldi.epfl.ch.

CBMS 91, Fourth IEEE Symp. on Computer-Based Medical Systems: May 12-14, 1991, Baltimore. Cosponsors: IEEE Computer Soc., IEEE Engineering in Medicine and Biology Soc., and IEEE Baltimore Section. Submit four copies of summary by **Dec. 5, 1990,** to Isaac Bankman, Johns Hopkins Univ., Applied Physics Lab., Bldg. 2-257, Johns Hopkins Rd., Laurel, MD 20723-6099, phone (301) 953-5000, ext. 4253, fax (301) 953-1093.

Eurographics 91, Conf. of the European Assoc. for Computer Graphics: Sept. 2-6, 1991, Vienna. Submit paper by **Dec. 10, 1990,** to Eurographics 91, c/o Interconvention, Austria Center Vienna, A-1450 Vienna, Austria, phone 43 (1) 2369-2640, fax 43 (1) 2369-648.

IJCAI 91, 12th Int'l Joint Conf. on Artificial Intelligence: Aug. 24-30, 1991, Sydney, Australia. Cosponsors: Australian Computer Soc. et al. Submit five copies of paper by **Dec. 10, 1990,** to John Mylopoulos or Ray Reiter, Computer Science Dept., Univ. of Toronto, Toronto, Ont., Canada M5S 1A4, phone (416) 978-5379, fax (416) 978-1455, e-mail ijcai@cs.toronto.edu.

DCC 91, Data Compression Conf.: Apr. 8-10, 1991, Snowbird, Utah. Sponsor: IEEE Computer Soc. TCC, NASA/CESDIS. Submit three copies of extended abstract by **Dec. 14, 1990,** to Martin Cohn, Computer Science Dept., Brandeis Univ., Waltham, MA 02254, phone (617) 736-2705, fax (617) 736-2741.

Fifth Israel Conf. on Computer Systems and Software Eng.: May 28-29, 1991, Herzlia, Israel. Sponsors: IEEE Computer Soc. Israeli Chapter et al. Submit four copies of extended abstract by **Dec. 14, 1990,** to M. Wino-kur, c/o ORTRA, PO Box 50432, Tel Aviv 61500, Israel, phone 972 (3) 664-825, fax 972 (3) 660-952.

Workshop on Parallel and Distributed Debugging: May 20-21, 1991, Santa Cruz, Calif. Cosponsors: ACM, US Navy Office of Naval Research. Submit 10 copies of extended abstract by **Dec. 15, 1990,** to Bart Miller, Computer Science Dept., Univ. of Wisconsin, 1210 W. Dayton St., Madison, WI 53706, phone (608) 263-3378, Internet bart@cs.wisc.edu.

Int'l J. of Microcomputer Applications seeks original research papers for its 1991 issues. Submit three copies of paper by **Dec. 15, 1990,** to Les Miller, Computer Science Dept., Iowa State Univ., Ames, IA 50011, phone (515) 294-4377, fax (515) 294-2574, e-mail lmiller@atanasoff.cs.iastate.edu.

SCAI 91, Third Scandinavian Conf. on Artificial Intelligence: May 21-24, 1991, Roskilde, Denmark. Submit three copies of abstract with long or short paper by **Dec. 20, 1990,** to Brian Mayoh, Computer Science Dept., Aarhus Univ., Ny Munkegade, Bldg. 540, DK-8000 Aarhus C, Denmark, fax 45 (86) 135725, e-mail brian@daimi.aau.dk.

Int'l J. of Computer-Aided VLSI Design plans a special issue on hardware accelerators for CAD. Publisher: Ablex. Submit five copies of full paper by **Dec. 31, 1990,** to Ausif Mahmood, Electrical Eng. Dept., Washington State Univ. at Tri-Cities, 100 Sprout Rd., Richland, WA 99352, phone (509) 375-9234, e-mail mahmood@prime.tricity.wsu.edu.

Second Int'l Conf. on Algebraic Methodology and Software Technology: May 22-24, 1991, Iowa City, Iowa. Submit abstract by **Jan. 1, 1991,** to AMAST Conf., Computer Science Dept., Univ. of Iowa, Iowa City, IA 52242.

IEEE Software is accepting entries for the fourth annual Gordon Bell Prize competition. Two $1,000 prizes will be awarded. Submit three- to four-page executive summary and full report by **Jan. 2, 1991,** to 1990 Gordon Bell Prize, c/o ***IEEE Software***, 10662 Los Vaqueros Cir., PO Box 3014, Los Alamitos, CA 90720-1264, Compmail soft.one, Internet: soft.one@compmail.com.

Second Physical Design Workshop: May 20-22, 1991, Laurel Highlands, Pa. Sponsor: ACM SIGDA. Submit 14 copies of the manuscript by **Jan. 7, 1991,** to Mary Jane Irwin, 333 Whitmore Lab, Computer Science Dept., Penn State Univ., University Park, PA 16802, e-mail mji@cs.psu.edu.

20th Int'l Conf. on Parallel Processing: Aug. 12-16, 1991, St. Charles, Ill. Sponsor: Pennsylvania State Univ. Submit paper by **Jan. 10, 1991,** to Kimming So, IBM Austin, Internal Zip 2812, 11400 Burnet Road, Austin, TX 78758 (on algorithms and applications); Herbert D. Schwetman, MCC, 3500 W. Balcones Center Dr., Austin, TX 78759 (on software); and Chuan-Lin Wu, Dept. of Electrical and Computer Eng., Univ. of Texas at Austin, Austin, TX 78712 (on hardware and other topics).

Compass 91, Sixth Conf. on Computer Assurance Systems Integrity, Software Safety, and Process Security: June 25-27, 1991, Gaithersburg, Md. Cosponsors: IEEE Aerospace and Electronics Soc., IEEE National Capital Area Council. Submit five copies of paper by **Jan. 11, 1991,** to Roger U. Fujii, Logicon, 222 W. Sixth St., San Pedro, CA 90731, phone (213) 831-0611, ext. 2420, e-mail r.fujii@ieee.org.

Sixth Int'l Conf. CAD/CAM, Robotics, and Factories of the Future: Aug. 19-22, 1991, London. Sponsor: Int'l Soc. for Productivity Enhancement. Submit three copies of abstract by **Jan. 11, 1991,** and manuscript by **June 1, 1991,** to H. Bera, Mechanical Eng. Dept., South Bank Polytechnic, 103 Borough Rd., London SE1 0AA, UK, phone 011 (91) 81-928-8989, ext. 2095, fax 011 (91) 81-261-9115.

Compsac 91, 15th Int'l Software and Applications Conf.: Sept. 11-13, 1991, Tokyo. Cosponsor: Information Processing Soc. of Japan. Submit six copies of paper by **Jan. 12, 1991,** to Lionel M. Ni, Michigan State Univ., Computer Science Dept., A714 Wells Hall, East Lansing, MI 48824-1027, phone (517) 353-4386, fax (517) 336-1061, Internet ni@cps.msu.edu (for the Americas, Europe, and Africa); or Motoei Azuma, Waseda Univ., c/o Business Center for Academic Societies of Japan, 3-23-1 Hongo, Bunkyo-ku, Tokyo 113, Japan, phone 81 (3) 817-5831, fax 81 (3) 817-5836.

ICANN 91, Int'l Conf. on Artificial Neural Networks: June 24-28, 1991, Espoo, Finland. Cosponsors: IEEE Neural Networks Council, Int'l Neural Network Soc. Submit manuscript by **Jan. 15, 1991,** to Olli Simula, Helsinki Univ. of Technology, SF-02150 Espoo, Finland, fax 358 (0) 451-3277, e-mail icann91@hutmc.hut.fi.

RIDT 91, Second Int'l Workshop on Raster Imaging and Digital Typography: Oct. 15-16, 1991, Boston. Cosponsor: Univ. of Massachusetts. Submittal deadline: **Jan. 15, 1991.** To obtain author information, contact Robert A. Morris, Mathematics and Computer Science Dept., Univ. of Massachusetts, Boston, MA 02125-3393, phone (617) 287-6466, e-mail ridt91-request@cs.umb.edu.

ESEC 91, Third European Software Eng. Conf.: Oct. 21-24, 1991, Milano, Italy. Spon-

sors: AFCET et al. Submit six copies of full paper and abstract by **Jan. 15, 1991,** to Alex van Lamsweerde, Unite d'Informatique, Univ. Catholique de Louvain, Place Sainte Barbe 2, B-1348 Louvain-La-Neuve, Belgium.

Third Int'l Workshop on Artificial Intelligence in Real-Time Control: Sept. 23-25, 1991, Sonoma Valley, Calif. Sponsor: Int'l Federation of Automatic Control. Submit four copies of extended abstract by **Jan. 15, 1991,** to Greg Suski, Lawrence Livermore Nat'l Lab, L-550, PO Box 808, Livermore, CA 94550, phone (415) 423-8070.

Sixth Int'l Workshop on Software Specification and Design: Oct. 25-26, 1991, Como, Italy. Submit five copies of regular or position paper by **Jan. 21, 1991,** to Carlo Ghezzi, Dip. di Elettronica Politecnico di Milano, Piazza Leonardo Da Vinci 32, 20133 Milano, Italia, e-mail reletr24@imipoli.bitnet.

IEEE Software plans a special issue on software for performance analysis. Submit six copies of manuscript by **Feb. 1, 1991,** to Kathleen Nichols, Apple Computer, 20525 Mariani, MS 76-3K, Cupertino, CA 95014, phone (408) 974-1136, e-mail nichols@apple.com; or Paul Oman, College of Eng., Univ. of Idaho, Moscow, ID 83843, phone (208) 885-6589, e-mail oman@ted.cs.uidaho.edu.

IEEE Trans. on Computers plans a special issue on artificial neural networks. Submit six copies of manuscript by **Feb. 1, 1991,** to Benjamin W. Wah, Coordinated Science Lab, MC228, Univ. of Illinois, 1101 W. Springfield Ave., Urbana, IL 61801-3082, phone (217) 333-3516, fax (217) 244-1764, e-mail wah%aquinas@uxc.cso.uiuc.edu.

ICCD 91, IEEE Int'l Conf. Symp. on Computer Design: Oct. 14-16, 1991, Cambridge, Mass. Cosponsors: IEEE Computer Soc. and IEEE Circuits and Systems Soc. Submit six copies of summary by **Feb. 1, 1991,** to Dwight Hill, AT&T Bell Labs, 3D-446, Murray Hill, NJ 07974, phone (201) 582-7766, e-mail dwight@research.att.com.

VLDB 91, 17th Int'l Conf. on Very Large Data Bases: Sept. 3-6, 1991, Barcelona, Spain. Sponsor: IEEE Computer Soc. Tech. Committee on Data Eng. et al. Submit five copies of paper by **Feb. 15, 1991,** to Amilcar Semadas, INESC, Rua Alves Redol, 9 ,7º, Apartado 10105, P-1017 Lisboa Codex, Portugal, e-mail inesc!acs%solo@relay.eu.net (US Internet), acs%solo@inesc.uucp (Europe Internet), or Guy M. Lohman, IBM Almaden Research Center, Dept. K55, Bldg. 801, 650 Harry Rd., San Jose, CA 95120-6099, e-mail lohman@ibm.com.

IEEE Infocom 92, 11th Conf. on Computer Communications: May 4-8, 1992, Florence, Italy. Cosponsor: IEEE Communications Soc. Submit six copies of paper by **June 30, 1991,** to L. Fratta, Politecnico di Milano, c/o Cefriel, Via Emanueli, 15, 20126 Milano, Italy, fax 39 (2) 2399-3587, e-mail fratta@imicefr.bitnet; or J. Kurose, Computer and Information Science Dept., Univ. of Massachusetts, Amherst, MA 01003, phone (413) 545-1585, e-mail kurose@cs.umass.edu.

November 1990

Cognitiva 90, Nov. 20-23, Madrid. Sponsor: AFCET. Contact Cognitiva 90, c/o Assoc. Francaise pour la Cybernetique Economique et Technique, 156 Bd. Pereire, 75017 Paris, France, phone 33 (1) 4766-2419, fax 33 (1) 4267-9312.

AI 90, Australian Joint Artificial Intelligence Conf., Nov. 21-23, Perth, Western Australia. Sponsor: Australian Computer Soc. Contact Les Kitchen, Univ. of Western Australia, Computer Science Dept., Nedlands, Western Australia, 6009, phone 61 (9) 380-2281, e-mail ai90@wacsvax.oz.au.

IFIP Workshop on Electronic Design Automation Frameworks, Nov. 26-28, Charlottesville, Va. Sponsor: Int'l Federation for Information Processing. Contact Ron Waxman, Univ. of Virginia, Thornton Hall, Charlottesville, VA 22903, phone (804) 924-6086.

IEEE 1990 Conf. on Software Maintenance, Nov. 26-29, San Diego, Calif. Contact Thomas M. Pigoski, 3890 Paradise Bay Dr., Gulfbreeze, FL 32561, phone (904) 932-4917.

NIPS 90, IEEE Conf. on Neural Information Processing Systems, Nov. 26-29, Denver. Contact Kathie Hibbard, Eng. Center, Univ. of Colorado, Campus Box 425, Boulder, CO 80309-0425.

Micro 23, 23rd Symp. and Workshop on Microprogramming and Microarchitecture, Nov. 27-29, Orlando, Fla. Cosponsor: ACM. Contact Chris Papachristou, Case Western Reserve Univ., Computer Eng. and Science Dept., Cleveland, OH 44106, phone (216) 368-5277, e-mail cap@alpha.ces.cwru.edu.

Iecon 90, 16th Conf. of the IEEE Industrial Electronics Soc., Nov. 27-30, Pacific Grove, Calif. Contact Robert Begun, 23609 Skyview Terr., Los Gatos, CA 95030, phone (408) 353-1560.

European Conf. on Hypertext, Nov. 27-30, Paris. Sponsor: INRIA. Contact Institut National de Recherche en Informatique et en Automatique, Service des Relations Exterieures, Domaine de Voluceau — Rocquencourt, BP 105, 78153 Le Chesnay Cedex, France; phone 33 (1)-39-63-55-00, fax 33 (1) 39-63-56-38.

Fifth Omni User Tech. Conf., Nov. 28, Itasca, Ill. Contact Omni User, PO Box A-3031, Chicago, IL 60690, phone (312) 470-8787.

IAPR Workshop on Machine Vision Applications, Nov. 28-30, Tokyo. Sponsor: Int'l Assoc. for Pattern Recognition. Contact Mikio Takagi, Inst. of Industrial Science, Univ. of Tokyo, 7-22-1 Roppongi, Minatoku, Tokyo 106, Japan, phone 81 (3) 479-0289, fax 81 (3) 423-2834, e-mail takagi@tkl.iis.u-tokyo.ac.jp.

IFIP Workshop on Silicon Architectures for Neural Nets, Nov. 28-30, St. Paul de Vence, France. Sponsor: Unisys. Contact Mariagiovanna Sami, Dip. di Elettronica, Politecnico di Milano, P.zza Leonardo da Vinci 32, I-20133 Milano, Italy, phone 39 (2) 2399-3516, fax 39 (2) 2399-3587.

Tools Pacific 90, Nov. 28-30, Sydney, Australia. Contact Myriam Wever, Victoire P/L, 8 Jane St., Balmain, NSW, Australia 2041, phone 61 (2) 810-2502, fax 61 (2) 810-3726.

December 1990

Fifth Int'l Conf. CAD/CAM, Robotics, and Factories of the Future, Dec. 2-5, Norfolk, Va. Sponsor: Int'l Soc. for Productivity Enhancement. Contact Alok Verma, Mechanical Eng. Tech. Dept., Old Dominion Univ., Norfolk, VA 23529, phone (804) 683-3765, fax (804) 683-4898.

First Int'l Symp. on Uncertainty and Analysis: Fuzzy Reasoning, Probabilistic Methods, and Risk Management, Dec. 3-5, College Park, Md. Sponsors: Univ. of Maryland et al. Contact Bilal M. Ayyub, Civil Eng. Dept., Univ. of Maryland, College Park, MD 20742, phone (301) 405-1956.

ACM SIGSoft 90, Fourth Symp. on Software Development Environments, Dec. 3-5, Irvine, Calif. Sponsor: ACM. Contact Dewayne E. Perry, AT&T Bell Labs, 600 Mountain Ave., Murray Hill, NJ 07974, phone (201) 582-2529.

Sixth Computer Security Applications Conf., Dec. 3-7, Tucson, Ariz. Sponsors: Am. Soc. for Industrial Security et al. Contact Marshall D. Abrams, Mitre Corp., 7525 Colshire Dr., M/S Z269, McLean, VA 22101, phone (703) 883-6938, e-mail abrams@mitre.org.

Tri-Ada 90, Dec. 3-7, Baltimore. Sponsor: ACM. Contact Erhard Ploedereder, Tartan Labs, 300 Oxford Dr., Monroeville, PA 15146, phone (412) 856-3600, fax (412) 856-3636, e-mail ploedere@tartan.com or ploedere@ajpo.sei.cmu.edu.

ICCV 90, Third Int'l Conf. on Computer Vision, Dec. 4-7, Osaka, Japan. Contact ICCV 90, IEEE Computer Soc., 1730 Massachusetts Ave. NW, Washington, DC 20036-1903, phone (202) 371-1013, fax (202) 728-0884.

CALENDAR

In the accompanying Calendar, the IEEE Computer Society logo identifies the conferences the society is sponsoring or participating in. Other conferences of interest to our readers, as well as their sponsors, are also listed.

For inclusion in Call for Papers or Calendar, submit the following information: event name, date(s), location, and sponsor(s) as well as the phone and fax numbers and the electronic address of the person to contact. In addition, for Calls for Papers listings, include the name of the person to whom papers should be submitted and the deadline for submittals.

Computer should receive the above-mentioned information at **least five weeks before the month of publication** (i.e., for the **January 1991** issue, send information for receipt by **November 20, 1990**) to Chuck Governale, Calendar Dept., *Computer*, PO Box 3014, Los Alamitos, CA 90720-1264.

RTSS 90, 11th Real-Time Systems Symp., Dec. 4-7, Orlando, Fla. Sponsor: IEEE Computer Soc. Tech. Committee on Real-Time Computing. Contact Doug Locke, IBM — MS 409, Systems Integration Div., 6600 Rockledge Dr., Bethesda, MD 20817, phone (301) 493-1496, e-mail cdl@cs.cmu.edu.

SEARCC 90, South East Asia Regional Computer Confederation Conf., Dec. 4-8, Manila, Philippines. Sponsor: Philippine Computer Soc. Contact Victor B. Gruet, Computer Information Systems, CIS Bldg., Meralco Compound, Ortigas Ave., 1602 Pasig, Metro Manila, Philippines, phone 63 (2) 722-1251, fax 63 (2) 722-0141.

CASE 90, Fourth Int'l Workshop on Computer-Aided Software Eng., Dec. 5-8, Irvine, Calif. Contact Elliott J. Chikofsky, Index Technology Corp, 1 Main St., Cambridge, MA 02142, phone (617) 577-8945, csnet chikofsky@northeastern.edu.

WSC 90, 1990 Winter Simulation Conf., Dec. 9-12, New Orleans. Contact Randall P. Sadowski, Systems Modeling Corp., 504 Beaver St., Sewickley, PA 15143, phone (412) 741-3727.

Second IEEE Symp. on Parallel and Distributed Processing, Dec. 9-13, Dallas. Cosponsor: IEEE Dallas Section. Contact Behrooz Shirazi, Computer Science Dept., Southern Methodist Univ., 6425 Airline Rd., Dallas, TX 75275-0122, phone (214) 692-2874, e-mail shirazi%smu.uucp@uunet.uu.net.

Workshop on Volume Visualization, Dec. 10-12, La Jolla, Calif. Cosponsor: ACM. Contact T. Todd Elvins, San Diego Supercomputing Center, Box 85608, San Diego, CA 92038, phone (619) 534-5128.

ICDT 90, Third Int'l Conf. on Database Theory, Dec. 12-14, Paris. Sponsor: INRIA. Contact INRIA, Domaine de Voluceau — Rocquencourt, BP 105, 78153 Le Chesnay Cedex, France, phone 33 (1) 3963-5500, fax 33 (1) 3963-5638.

High-Integrity Systems Lecture Series, Dec. 17, Gaithersburg, Md. Sponsor: Nat'l Inst. of Standards and Technology. Contact Dolores Wallace, NIST, Bldg., B266, Gaithersburg, MD 20899, phone (301) 975-3340, e-mail wallace@swe.swe.ncsl.nist.gov.

10th Conf. on Foundations of Software Technology and Theoretical Computer Science, Dec. 17-19, Bangalore, India. Contact Y.N. Srikant, Indian Inst. of Science, Bangalore 560 012, India, phone (812) 334-411.

1990 IEEE Workshop on Languages and Architectures for Automation, Dec. 19-21, Honolulu. Sponsors: Pacific Int'l Center for High Technology Research et al. Contact D.Y.Y. Yun, Univ. of Hawaii, 711 Kapiolani Blvd., Suite 200, Honolulu, HI 96813-5249, phone (808) 539-1532, fax (808) 941-1399; or Shi-Kuo Chang, 322 Alumni Hall, Univ. of Pittsburgh, Pittsburgh, PA 15260, phone (412) 624-8493, fax (412) 624-8465, e-mail chang@vax.cs.pitt.edu.

Seventh Israeli Conf. on Artificial Intelligence and Computer Vision, Dec. 26-27, Tel Aviv, Israel. Contact A. Bruckstein, Faculty of Computer Science, Technion, 32000 Haifa, Israel, e-mail freddy@techsel.bitnet; or Shmuel Peleg, David Sarnoff Research Center, CN 5300, Princeton, NJ 08543-5300, phone (609) 734-2284, e-mail peleg@vision.sarnoff.com.

January 1991

Fourth CSI/IEEE Int'l Symp. on VLSI Design, Jan. 5-8, New Delhi, India. Sponsors: Computer Soc. of India et al. Contact Yashwant K. Malaiya, Computer Science Dept., Colorado State Univ., Fort Collins, CO 80523, phone (303) 491-7031, fax (303) 491-2293, e-mail malaiya@ravi.cs.colostate.edu; or D. Roy Chowdhury, Gateway Design Automation, SDF#A-1, Noida Export Processing Zone, PO NEPZ, Noida 201305, India, phone 91 (05736) 62342, fax 91 (05736) 62343.

SIAM Workshop on Automatic Differentiation of Algorithms, Jan. 7-9, Breckenridge, Colo. Contact Soc. for Industrial and Applied Math., Conf. Coordinator, Dept. CC0590, 3600 University City Science Center, Philadelphia, PA 19104-2688, phone (215) 382-9800, fax (215) 386-7999, e-mail siamconfs@wharton.upenn.edu.

Int'l Workshop on Formal Methods in VLSI Design, Jan. 9-11, Puerto Rico. Cosponsors: ACM, IFIP. Contact P.A. Subrahmanyam, Rm. 4F-530, AT&T Bell Labs, Holmdel, NJ 07733, phone (201) 949-5812, fax (201) 949-3697, e-mail subra@vax135.att.com.

Fifth Tech. Conf. on the X Window System, Jan. 14-16, Boston. Sponsor: MIT X Consortium. Contact X Tech. Conf, Rm. 217, MIT Lab for Computer Science, 545 Technology Square, Cambridge, MA 02139.

Int'l Conf. on Multimedia Information Systems, Jan. 16-18, Singapore. Contact Desai Narasimhalu or Juzar Motiwalla, Inst. of Systems Science, Nat'l Univ. of Singapore, Heng Mui Keng Terr., Kent Ridge, Singapore 0511, phone (65) 772-2075, fax (65) 772-2002, Bitnet issad@nusvm.

Int'l Workshop on Unix-Based Software Development Environments, Jan. 16-18, Dallas. Sponsor: Usenix Assoc. Contact Use-nix Conf. Office, 22672 Lambert St., Suite 613, El Toro, CA 92630, phone (714) 588-8649.

Conf. on Optics, Electro-Optics, and Laser Applications in Science and Eng., Jan. 20-25, Los Angeles. Sponsor: Int'l Soc. for Optical Eng. Contact SPIE, PO Box 10, Bellingham, WA 98227-0010, phone (206) 676-3290, fax (206) 647-1445.

PADS, Workshop on Parallel and Distributed Simulation, Jan. 21-23, Anaheim, Calif. Cosponsors: ACM, SCS. Contact David M. Nicol, Computer Science Dept., College of William and Mary, Williamsburg, VA 23185, phone (804) 221-3458, e-mail nicol@cs.wm.edu.

Winter 91 Unix Tech. Conf., Jan. 21-15, Dallas. Sponsor: Usenix Assoc. Contact Usenix Conf. Office, 22672 Lambert St., Suite 613, El Toro, CA 92630, phone (714) 588-8649.

Second ACM-SIAM Symp. on Discrete Algorithms, Jan. 28-30, San Francisco. Contact SIAM Conf. Coordinator, Dept. CC0590, 3600 University City Science Center, Philadelphia,

PA 19104-2688, phone (215) 382-9800, fax (215) 386-7999, e-mail siamconfs@wharton.upenn.edu.

IEEE Int'l Conf. on Wafer Scale Integration, Jan. 29-31, San Francisco. Cosponsors: IEEE Components, Hybrids, and Manufacturing Technology Soc. Contact Terry Chappell, 730 Encino Dr., Aptos, CA 95003, phone (408) 662-1936; or R. Mike Lea, Brunel Univ., Uxbridge UB8 3PH, UK, phone (44) 895-74000, ext. 2821, fax (44) 895-58728, e-mail mike.lea@brunel.ac.uk.

February 1991

Systems/USA Technology Conf., Feb. 11-13, Anaheim, Calif. Sponsor: Am. Electronics Assoc. Contact AEA, 5201 Great America Pkwy., Santa Clara, CA 95054, phone (503) 359-5873 or (408) 987-4204, fax (503) 357-3839 or (408) 970-8565.

Fifth Int'l Conf. on Modeling Techniques and Tools for Computer Performance Evaluation, Feb. 13-15, Torino, Italy. Contact Maria Carla Calzarossa, Dip. di Informatica e Sistemistica, Univ. di Pavia, Via Abbiategrasso, 209, 27100 Pavia, Italy, phone 39 (382) 391-350, fax 39 (382) 422-881, e-mail mcc@ipvpel.infn.it.

ISSCC 91, 1991 IEEE Int'l Solid-State Circuits Conf., Feb. 13-15, San Francisco. Sponsors: IEEE Solid-State Circuits Council et al. Contact Diane Suiters, Courtesy Associates, 655 15th St. NW, Suite 300, Washington, DC 20005, phone (202) 639-4255.

PCCS 1, First Int'l Workshop on Performability Modeling of Computer and Comm. Systems, Feb. 18-19, Enschede, The Netherlands. Contact Nico M. van Dijk, Free Univ., Faculty of Economics, PO Box 7161, 1007 MC Amsterdam, The Netherlands, phone 31 (20) 548-7061, fax 31 (20) 462-645, e-mail ectricvu@sara.nl.

CAIA 91, Seventh IEEE Conf. on Artificial Intelligence Applications, Feb. 24-28, Miami Beach, Fla. Contact IEEE Computer Soc., 1730 Massachusetts Ave. NW, Washington, DC 20036-1903, phone (202) 371-1013.

Fourth Topical Meeting on Robotics and Remote Systems for Hazardous Environments, Feb. 24-28, Albuquerque, N.M. Contact Raymond W. Harrigan, Div. 1414, Sandia Nat'l Labs, Albuquerque, NM 87185, phone (505) 846-6278, fax (505) 846-7425.

EDAC 91, European Design Automation Conf., Feb. 25-28, Amsterdam. Sponsor: Institution of Electrical Engineers. Contact Secretariat, EDAC 91, CEP Consultants, 26-28 Albany St., Edinburgh EH1 3QH, Scotland, phone 44 (31) 557-2478, fax 44 (31) 557-5749.

Compcon Spring 91, Feb. 25-Mar. 1, San Francisco. Contact Compcon Spring 91, IEEE Computer Soc., 1730 Massachusetts Ave. NW, Washington, DC 20036-1903, phone (202) 371-1013.

March 1991

Fifth Int'l Workshop on High-Level Synthesis, Mar. 3-6, Buhlerhohe, Germany. Cosponsors: IEEE et al. Contact Raul Camposano, IBM T.J. Watson Research Center, PO Box 218, Yorktown Heights, NY 10598, phone (914) 945-3871, e-mail raulc@ibm.com.

22nd Tech. Symp. on Computer Science Education, Mar. 7-8, San Antonio, Texas. Sponsor: ACM SIGSCE. Contact Nell Dale, Computer Science Dept., Univ. of Texas at Austin, Austin, TX 78712, phone (512) 471-9539, e-mail ndale@cs.utexas.edu.

Fourth Computer Virus and Security Conf., Mar. 14-15, New York City. Sponsor: Data Processing Management Assoc. Financial Industries. Contact Judy S. Brand, PO Box 6313, FDR Station, New York, NY 10150, phone (800) 835-2246.

Third IEE Conf. on Telecomm., Mar. 17-20, Edinburgh, Scotland. Sponsor: Institution of Electrical Engineers. Contact Conf. Services, IEE, Savoy Place, London WC2R 0BL, UK, phone 44 (71) 240-1871, fax 44 (71) 240-7735.

Symp. on Experiences with Distributed and Multiprocessor Systems, Mar. 21-22, Atlanta. Sponsor: Usenix Assoc. Contact George Leach, AT&T Paradyne, MS LG-129, PO Box 2826, Largo, FL 34649-2826, phone (813) 530-2376, e-mail reggie@pdn.paradyne.com.

Fifth SIAM Conf. on Parallel Processing and Scientific Computing, Mar. 25-27, Houston. Contact Soc. for Industrial and Applied Math. Conf. Coordinator, Dept. CC0590, 3600 University City Science Center, Philadelphia, PA 19104-2688, phone (215) 382-9800, fax (215) 386-7999, e-mail siamconfs@wharton.upenn.edu.

Advanced Research in VLSI Conf., Mar. 25-27, Santa Cruz, Calif. Sponsors: Univ. of California at Santa Cruz, Univ. of California at Berkeley. Contact Kevin Karplus, Computer Engineering, Univ. of California at Santa Cruz, Santa Cruz, CA 95064, Internet karplus@ce.ucsc.edu.

Auto Carto 10, 10th Int'l Symp. on Automated Cartography, Mar. 25-28, Baltimore. Cosponsors: Am. Cartographic Assoc. et al. Contact ACSM/ASPRS/Auto Carto 10, 5410 Grovesnor Lane, Bethesda, MD 20814.

CEEDA 91, Int'l Conf. on Concurrent Eng. and Electronic Design Automation, Mar. 26-28, Bournemouth, Dorset, UK. Sponsors: Institution of Electrical Engineers et al. Contact Sa'ad Medhat, Bournemouth Polytechnic, Poole House, Talbot Campus, Fern Barrow, Dorset BH12 5BB, UK, phone 44 (81) 595-492, fax 44 (81) 595-368, e-mail saiad medhat@eurolom.ie.

10th IEEE Int'l Phoenix Conf. on Computers and Comm., Mar. 27-30, Scottsdale, Ariz. Sponsors: IEEE, IEEE Comm. Soc. Contact Oris Friesen, Bull HN, PO Box 8000, MS A93, Phoenix, AZ 85066, phone (602) 862-5200, e-mail friesen@system-m.phx.bull.com.

April 1991

24th Computer Simulation Conf., Apr. 1-5, New Orleans. Sponsor: Soc. for Computer Simulation. Contact George W. Zobrist, Computer Science Dept., Univ. of Missouri at Rolla, Rolla, MO 65401, phone (314) 341-4836, e-mail c2816@umrvmb.umr.edu.

Dasfaa 91, Second Int'l Symp. on Database Systems for Advanced Applications, Apr. 2-4, Tokyo. Sponsor: Information Processing Soc. of Japan. Contact Yahiko Kambayashi, Computer Science Dept., Kyushu Univ., 6-10-1 Hakozaki, Higashi Fukuoka 812, Japan, phone 81 (92) 641-1101, ext. 5407; or Yoshifumi Masunaga, Univ. of Library and Information Science, 1-2 Kasuga, Tsukuba, Ibaraki 305, Japan, phone 81 (298) 52-0511, ext. 340, fax 81 (298) 52-4326, e-mail masunaga@ulis.ac.jp.

Flairs 91, Florida Artificial Intelligence Research Symp., Apr. 2-5, Cocoa Beach, Fla. Sponsor: Florida Artificial Intelligence Research Soc. Contact Avelino J. Gonzalez, Computer Eng. Dept., Univ. of Central Florida, Orlando, FL 32816, phone (407) 281-5027, e-mail fdgonzal%ucf1vm.bitnet@cunyvm.cuny.edu.

SAC 91, 1991 Symp. on Applied Computing, Apr. 3-5, Kansas City, Mo. Sponsor: Univ. of Missouri — Kansas City. Contact Richard G. Hetherington, SAC 91, Univ. of Missouri — Kansas City, Computer Science Telecommunications Program, 5100 Rockhill Rd., Kansas City, MO 64110-2499, phone (816) 235-2399.

Third Symp. on Integrated Ferroelectrics, Apr. 3-5, Colorado Springs, Colo. Contact Conf. Secretary, Microelectronics Research Lab, Univ. of Colorado at Colorado Springs, PO Box 7150, Colorado Springs, CO 80933-7150, phone (719) 593-3488, fax (719) 594-4257.

Computer Graphics and Education 91, Apr. 4-6, Barcelona, Spain. Sponsor: Int'l Federation for Information Processing. Contact Steve Cunningham, Computer Science Dept, California State Univ. at Stanislaus, Turlock, CA 95380, phone (209) 667-3176, e-mail rsc@altair.csustan.edu; or Roger Hubbold, Computer Science Dept., Univ. of Manchester, Oxford Road, Manchester M13 9PL, UK, e-mail hubbold@uk.ac.man.cs.

IEEE Infocom 91, Conf. on Computer Comm., Apr. 7-11, Miami, Fla. Cosponsors: IEEE Computer Soc. and Comm. Soc. Contact N. Shacham, IEEE Infocom 91, SRI Int'l, 333 Ravenswood Ave., Menlo Park, CA 94025, phone (415) 859-5710, e-mail shacham@sri.com.

1991 IEEE Int'l Conf. on Robotics and Automation, Apr. 7-12, Sacramento, Calif. Sponsor: IEEE Robotics and Automation Soc. Contact Robotics and Automation, PO Box 3216, Silver Spring, MD 20918, phone (301) 434-1990.

IMS 91, First IEEE Int'l Workshop on Interoperability in Multidatabase Systems, Apr. 8-9, Kyoto, Japan. Contact Ahmed K. Elmagarmid, Purdue Univ., Computer Sciences Dept., West Lafayette, IN 47907, phone (317) 494-1998; or Yutaka Matsushita, Instrumentation Dept., Keio Univ., Hiyoshi, Yokohama, Japan, phone 81 (44) 63-1141, ext. 3564.

DCC 91, Data Compression Conf., Apr. 8-10, Snowbird, Utah. Sponsor: IEEE Computer Soc. TCC, NASA/CESDIS. Contact John H. Reif, Computer Science Dept., Duke Univ., Durham, NC 27707, phone (919) 660-6568.

ASPLOS 4, Fourth Int'l Conf. on Architectural Support for Programming Languages and Operating Systems, Apr. 8-11, Santa Clara, Calif. Sponsor: ACM. Contact Bob Rau, Hewlett-Packard Labs, 1501 Page Mill Rd., Bldg. 3U, Palo Alto, CA 94304, fax (415) 857-8558, e-mail rau@hplabs.hp.com.

Seventh Int'l Conf. on Data Eng., Apr. 8-12, Kobe, Japan. Contact Ming T. (Mike) Liu, Computer and Information Science Dept., Ohio State Univ., 2036 Neil Ave., Columbus, OH 43210-1277, phone (614) 292-1837, e-mail liu@cis.ircc.ohio-state.edu; or Data Eng. 91, IEEE Computer Soc., 1730 Massachusetts Ave. NW, Washington, DC 20036-1903, phone (202) 371-1013, fax (202) 728-0884.

IFIP Working Conf. on Modeling in Computer Graphics, Apr. 8-12, Tokyo. Sponsor: IFIP TC 5/WG 5.10. Contact Tosiyasu L. Kunii, Information Science Dept., Faculty of Tokyo, Univ. of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo 113, Japan, phone 81 (3) 816-1783, fax 81 (3) 818-4607, e-mail b39756@tansei.cc.u-tokyo.ac.jp.

ETC 91, 1991 European Test Conf., Apr. 10-12, Munich, Germany. Sponsor: VDE (Zentralstelle Tagungen und Seminare). Contact Peter Stilke, VDE, Stresemannallee 15, D-6000 Frankfurt 70, Germany, phone (69) 6308-203, fax (69) 6308-273.

RTA 91, Fourth Int'l Conf. on Rewriting Techniques and Applications, Apr. 10-12, Como, Italy. Sponsor: State Univ. of Milan. Contact G. Degli Antoni or Marelva Bianchi, Dip. di Scienze Dell' Informazione, Univ. di Milano, Via Milano Moretto da Brescia 9, I-20133 Milano, Italia, phone 39 (02) 7575-201, fax 39 (02) 7611-0556, e-mail gdantoni@imisiam.bitnet.

Ninth IEEE VLSI Test Symp., Apr. 16-18, Atlantic City, N.J. Cosponsor: IEEE Philadelphia Section. Contact Mukund Modi, Naval Air Eng. Center, ATE Software Center, Code: 52514, Lakehurst, NJ 08733, phone (201) 323-7002, fax (201) 323-7445.

CHDL 91, 10th Int'l Symp. on Computer Hardware Description Languages and their Applications, Apr. 22-24, Marseille, France. Cosponsors: Int'l Federation for Information Processing et al. Contact Dominique Borrione, Imag/Artemis, BP 53X, 38041 Grenoble Cedex, France, phone (33) 7651-4604, ext. 5240, fax (33) 7651-9637, e-mail borrione@imag.imag.fr.

Second European Distributed Memory Computing Conf., Apr. 22-24, Munich, Germany. Cosponsors: Gesellschaft fur Informatik et al. Contact Arndt Bode, Computer Science, Technische Univ. Munich, POB 20-24-20, D-8000 Munich 2, Germany, phone 49 (89) 2105-8240, e-mail bode@infovax.informatik.tu-muenchen.dbp.de.

KMET 91, 1st'l Conf. on Knowledge Modeling and Expertise Transfer, Apr. 22-24, Sophia Antipolis, France. Cosponsors: Assoc. Francaise pour la Cybernetique Economique et Technique et al. Contact KMET 91, Univ. de Nice, Sophia Antipolis, CNRS, 13S-LISAN, Daniele Herin-Aime, bat. 4, rue A. Einstein, 06560, Valbonne, France, phone (33) 92-94-26-27, fax (33) 92-94-28-98, e-mail dh@cerisi.cerisi.fr.

Second Int'l Conf. on Systems Integration, Apr. 22-25, Morristown, N.J. Cosponsors: New Jersey Inst. of Technology et al. Contact Peter A. Ng, Computer and Information Science Dept., New Jersey Inst. of Technology, University Heights, Newark, NJ 07102, phone (201) 596-3387, e-mail ng_p@vienna.njit.edu.

NCGA 91, 1991 Nat'l Computer Graphics Assoc. Conf., Apr. 22-25, Chicago. Contact NCGA, 2722 Merrilee Dr., Suite 200, Fairfax, VA 22031, phone (703) 698-9600.

CHI 91, 1991 Conf. on Human Factors in Computing Systems, Apr. 27-May 2, New Orleans. Sponsor: ACM. Contact Keith Butler, Boeing, Advanced Technology Center, PO Box 24346, M/S 7L-64, Seattle, WA 98124, phone (206) 865-3389; or June Davis, 13 Annapolis St., Annapolis, MD 21401, phone (301) 269-6801.

Int'l Conf. on Cognitive Sciences, Apr. 29-May 2, Montreal. Cosponsors: AFCET et al. Contact Gilles Gauthier, Math. and Computer Science Dept., Univ. of Quebec, PO Box 8888, Station A, Montreal, Que., Canada H3C 3P8, phone (514) 987-8212, fax (514) 987-8477.

Fifth Int'l Parallel Processing Symp., Apr. 30-May 2, Anaheim, Calif. Sponsor: IEEE Computer Soc. Orange County Chapter. Contact Larry H. Canter, Computer Systems Approach, 1140 S. Raymond Ave., Suite B, Fullerton, CA 92631, phone (714) 738-3414, fax (714) 738-3470.

May 1991

Fifth SIAM Int'l Symp. on Domain Decomposition Methods for Partial Differential Equations, May 6-8, Norfolk, Va. Contact Soc. for Industrial and Applied Math., Dept. CC0590, 3600 University City Science Center, Philadelphia, PA 19104-2688, phone (215) 382-9800, fax (215) 386-7999, e-mail siamconfs@wharton.upenn.edu.

SID 91, 1991 Int'l Symp., Seminar, and Exhibition, May 6-10, Anaheim, Calif. Sponsor: Soc. for Information Display. Contact SID, c/o Palisades Inst. for Research Services, 201 Varick St., Suite 1140, New York, NY 10014, phone (212) 620-3371, fax (212) 620-3379.

CBMS 91, Fourth IEEE Symp. on Computer-Based Medical Systems, May 12-14, Baltimore. Cosponsors: IEEE Computer Soc., IEEE Engineering in Medicine and Biology Soc., and IEEE Baltimore Section. Contact Jeffery C. Lesho, Johns Hopkins Univ., Applied Physics Lab., Bldg. 2-257, Johns Hopkins Rd., Laurel, MD 20723-6099, phone (301) 953-5000, ext. 8057.

CICC 91, IEEE Custom Integrated Circuits Conf., May 12-15, San Diego, Calif. Contact Jim Lipman, VLSI Technology, 1109 McKay Dr. MS-32, San Jose, CA 95131, phone (408) 434-7673.

44th Conf. of the Soc. for Imaging Science and Technology, May 12-17, St. Paul, Minn. Contact SPSE, 7003 Kilworth Lane, Springfield, VA 22151, phone (703) 642-9090, fax (703) 642-9094.

ICSE 13, 13th Int'l Conf. on Software Eng., May 13-17, Austin, Texas. Cosponsor: ACM. Contact ICSE 13, Bryan Fugate, MCC, 3500 W. Balcones Center Dr., Austin, TX 78759-6509, phone (512) 338-3330; MCC, PO Box 200015, Austin, TX 78720-0015; or ICSE 13, IEEE Computer Soc., 1730 Massachusetts Ave. NW, Washington, DC 20036-1903, phone (202) 371-1013.

CompEuro 91, IEEE Int'l Conf. on Advanced Computer Technology, Reliable Systems, and Applications, May 13-17, Bologna, Italy. Cosponsors: IEEE Region 8 et al. Contact Vito Monaco, Dip. Eletronica Informatica E Sistemistica, Univ. Di Bologna, Viale Risorgimento, I-60136, Bologna, Italy.

Ada-Europe Athens 91 Conf., May 13-17, Athens. Cosponsors: Ada-Europe et al. Contact Z. Kaplanidis, Zita Tourist Club, 46, Voulis St., GR - 10558 Athens, Greece, phone 30 (1) 323-9744/7, fax 30 (1) 324-1720.

CCW 91, Third IEEE Conf. on Computer Workstations, May 15-17, Falmouth, Mass. Sponsor: IEEE Computer Soc. Tech. Committee on Operating Systems. Contact Luis-Felipe Cabrera, IBM Almaden Research Center, MC K55/801, 650 Harry Rd., San Jose, CA 95120-6099, phone (408) 927-1838, e-mail cabrera@ibm.com; or Kenneth Kane, Boston Development Center, Sun Microsystems, 2 Federal St., Billerica, MA 01802, phone (508) 671-0367, e-mail kkane@east.sun.com.

North Am. Fuzzy Information Processing Soc. Workshop, May 15-17, Columbia, Mo. Contact Jim Keller, Electrical and Computer

Eng., Univ. of Missouri — Columbia, Columbia, MO 65211, phone (314) 882-7339, fax (314) 882-0397.

Int'l Symp. on Software Reliability Eng., May 17-18, Austin, Texas. Co-sponsors: IEEE Computer Soc. Tech. Committee on Software Eng. and the Nat'l Aeronautics and Space Administration. Contact Anneliese von Mayrhauser, Computer Science Dept., Illinois Inst. of Technology, 600 S. Lambert Rd., Glen Elyn, IL 60137, phone (708) 790-4385, e-mail csavm@karl.iit.edu.

Workshop on Parallel and Distributed Debugging, May 20-21, Santa Cruz, Calif. Co-sponsors: ACM, US Navy Office of Naval Research. Contact Bart Miller, Computer Science Dept., Univ. of Wisconsin, 1210 W. Dayton St., Madison, WI 53706, phone (608) 263-3378, Internet bart@cs.wisc.edu.

1991 IEEE Symp. on Research in Security and Privacy, May 20-22, Oakland, Calif. Sponsor: IEEE Computer Soc. Tech. Committee on Security and Privacy. Contact Daniel Schnackenberg, Boeing, MS 9P-64, PO Box 3999, Seattle, WA 98124, phone (206) 657-5595, e-mail schnackenberg@dockmaster.ncsc.mil.

Second Physical Design Workshop, May 20-22, Laurel Highlands, Pa. Sponsor: ACM SIGDA. Contact Antun Domic, HLO2-3J3, DEC, 77 Reed Rd., Hudson, MA 01749, e-mail domic@cadsys.dec.com.

ICDCS 91, 11th Int'l Conf. on Distributed Computing Systems, May 20-24, Arlington, Texas. Contact Bill D. Carroll, Computer Science Eng. Dept., Univ. of Texas at Arlington, Box 19015, Arlington, TX 76019-0015, phone (817) 273-3785, e-mail carroll@evax.ari.utexas.edu.

SESAW, Fourth Software Eng. Standard Application Workshop, May 21-23, San Diego, Calif. Contact Vera V. Edelstein, Nynex, 500 Westchester Ave., White Plains, NY 10604, phone (914) 683-2888.

SCAI 91, Third Scandinavian Conf. on Artificial Intelligence, May 21-24, Roskilde, Denmark. Contact Brian Mayoh, Computer Science Dept., Aarhus Univ., Ny Munkegade, Bldg. 540, DK-8000 Aarhus C, Denmark, phone 45 (86) 127188, fax 45 (86) 135725, e-mail brian@daimi.aau.dk.

Second Int'l Conf. on Algebraic Methodology and Software Technology, May 22-24, Iowa City, Iowa. Contact Teodor Rus, Computer Science Dept., Univ. of Iowa, Iowa City, IA 52242, phone (319) 335-0694, e-mail rus@herky.cs.uiowa.edu.

Melecon 91, Fifth Mediterranean Electrotech. Conf., May 22-24, Ljubljana, Yugoslavia. Cosponsors: IEEE Region 8 Yugoslavia Section et al. Contact Melecon 91 Secretariat, Fakulteta za elektrotehniko, Trzaska 25, 61001 Ljubljana, Yugoslavia, phone 38 (61) 265-161, fax 38 (61) 264-990.

Computer Animation 91, May 22-25, Geneva, Switzerland. Cosponsors: Computer Graphics Soc. Contact Nadia M. Thalmann, Mira Lab CUI, Univ. of Geneva, 12 rue du Lac, CH 1207, Geneva, Switzerland, phone 41 (22) 787-6581, fax 41 (22) 735-3905, e-mail thalmann@uni2a.unige.ch.

21st Int'l Symp. on Multiple-Valued Logic, May 26-29, Victoria, Canada. Contact D.M. Miller, Computer Science Dept., Univ. of Victoria, PO Box 1700, Victoria, B.C., Canada, V8W 2Y2, phone (604) 721-7220, fax (604) 721-7292, e-mail dmill@csr.uvic.edn.

ISCA 18, 18th Int'l Symp. on Computer Architecture, May 26-30, Toronto. Cosponsor: ACM. Contact K.C. Smith, Univ. of Toronto, Electrical Eng. Dept., Toronto, Ont. M5S 1A4, Canada, phone (416) 978-5033.

ICCI 91, Int'l Conf. on Computing and Information, May 27-29, Ottawa, Canada. Sponsors: Carleton Univ, Ottawa; Natural Sciences and Eng. Research Council of Canada. Contact Frank Fiala, School of Computer Science, Carleton Univ., Ottawa, Canada K1S 5B6, phone (613) 788-4333, fax (613) 788-4334, e-mail icci@scs.carleton.ca.

Fifth Israel Conf. on Computer Systems and Software Eng., May 28-29, Herzlia, Israel. Sponsors: IEEE Computer Soc. Israeli Chapter et al. Contact M. Winokur, c/o ORTRA, PO Box 50432, Tel Aviv 61500, Israel, phone 972 (3) 664-825, fax 972 (3) 660-952.

June 1991

Workshop on Directions in Automated CAD-Based Vision, June 2-3, Maui, Hawaii. Contact Linda Shapiro, Computer Science and Eng. Dept., FR-35, Univ. of Washington, Seattle, WA 98195, phone (206) 543-2196, fax (206) 543-3842.

Fourth Int'l Conf. on Industrial and Eng. Applications of Artificial Intelligence and Expert Systems, June 2-5, Kauai, Hawaii. Sponsors: ACM et al. Contact Moonis Ali, Univ. of Tennessee Space Inst., MS15, B.H. Goethert Pkwy., Tullahoma, TN 37388-8897, phone (615) 455-0631, ext. 236, fax (615) 454-2354, e-mail alif@utsiv1.bitnet.

11th Int'l Conf. on Decision-Support Systems, June 3-5, Manhattan Beach, Calif. Sponsor: Inst. for Management Sciences. Contact TIMS, 290 Westminster St., Providence, RI 02903.

CVPR 91, IEEE Computer Soc. Conf. on Computer Vision and Pattern Recognition, June 3-7, Lahaina, Maui, Hawaii. Contact Shahriar Negahdaripour, Electrical Eng. Dept., Univ. of Hawaii at Manoa, 2540 Dole St., Honolulu, HI 96822, e-mail shahriar@wiliki.eng.hawaii.edu.

20th Mumps Users Group Meeting, June 3-7, New Orleans. Contact Mumps Users Group, 4321 Hartwick Rd., Suite 100, College Park, MD 20740, phone (301) 779-6555, fax (301) 779-7674.

Symp. on Solid Modeling Foundations and CAD/CAM Applications, June 5-7, Austin, Texas. Sponsor: ACM SIGGraph. Contact Joshua Turner, CII 7015, RDRC, Rensselaer Polytechnic Inst., Troy, NY 12180-3590, phone (518) 276-6751, fax (518) 276-2702, e-mail jturner@rdrc.rpi.edu.

Parle 91, Conf. on Parallel Architectures and Languages Europe, June 10-13, Eindhoven, The Netherlands. Cosponsors: Commission of European Communities et al. Contact F. Stoots, Philips Research Labs. PO Box 80.000, 5600 JA Eindhoven, The Netherlands, fax 31 (40) 744-758, e-mail stoots@dooma.prl.philips.nl.

ISCAS 91, 24th IEEE Int'l Symp. on Circuits and Systems, June 11-14, Singapore. Sponsor: IEEE Circuits and Systems Soc. Contact ISCAS 91 Secretariat, Comm. Int'l Associates, 44/46 Tanjong Pagar Rd., Singapore 0208, phone (65) 226-2823, fax (65) 226-2877.

SCM 3, Third Int'l Software Configuration Management Workshop, June 12-14, Trondheim, Norway. Cosponsors: ACM, et al. Contact Reidar Conradi, Computer Systems and Telematics Div., Norwegian Inst. of Technology, N-7034 Trondheim, Norway, phone 47 (7) 593-444; or Peter Feiler, Software Eng. Inst., Carnegie Mellon Univ., Pittsburgh, PA 15213-3890, phone (412) 268-7790, e-mail phf@sei.cmu.edu.

1991 ACM Symp. on Personal and Small Computers, June 12-14, Toronto. Cosponsors: Nat'l Research Council of Canada et al. Contact Michael Bauer, Computer Science Dept., Middlesex College, Univ. of Western Ontario, London, Ont., Canada N6A 5B7, e-mail bauer@csd.uwo.ca or bauer@uwovax.bitnet.

DAC 91, 28th ACM/IEEE Design Automation Conf., June 16-21, San Francisco. Contact MP Associates, 7490 Clubhouse Rd., Suite 102, Boulder, CO 80301, phone (303) 530-4333.

1991 ACM Int'l Conf. on Supercomputing, June 17-21, Cologne, Germany. Cosponsors: Gesellschaft für Informatik et al. Contact Ruediger Esser, FKA-ZAM, D-5170 Juelich, Germany, phone 49 (2461) 61-6588, fax 49 (2461) 61-6656, e-mail zdv003@djukfa11.bitnet.

Eighth Int'l Conf. on Testing Computer Software, June 17-21, Washington, DC. Sponsor: Data Processing Management Assoc. Educational Foundation. Contact Genevieve Houston-Ludlam, Frontier Technologies, 190 Admiral Cochran Dr., Suite 180, Annapolis, MD 21401, phone (301) 266-8244, fax (301) 224-3840.

CGI 91, Int'l Conf. on Computer Graphics, June 22-28, Cambridge, Mass. Cosponsors: Computer Graphics Soc., MIT. Contact Barbara Dullea, Ocean Eng. Dept., MIT Rm. 5-435, 77 Massachusetts Ave., Cambridge, MA 02139, fax (617) 253-8125, e-mail barbara@deslab.mit.edu.

1991 IEEE Int'l Symp. on Information Theory, June 23-29, Budapest, Hungary. Contact Anthony Ephremides, Electrical Eng. Dept., Univ. of Maryland, College Park, MD 20742, phone (301) 454-6871, e-mail tony@eng.umd.edu.

ICANN 91, Int'l Conf. on Artificial Neural Networks, June 24-28, Espoo, Finland. Co-sponsors: IEEE Neural Networks Council, Int'l Neural Network Soc. Contact Congress Management Systems, PO Box 151, SF-00141 Helsinki, Finland, phone 358 (0) 175-355, fax 358 (0) 170-122.

FTCS 21, 21st Int'l Symp. on Fault-Tolerant Computing, June 25-27, Montreal. Sponsor: IEEE Computer Soc. Tech. Committee on Fault-Tolerant Computing. Contact Vinod K. Agarwal, McGill Univ., Electrical Eng. Dept., 3480 University St., Montreal, Que., Canada H3A 2A7, phone (514) 398-7136, fax (514) 398-4470, e-mail agarwal@spock.ee.mcgill.ca.

Compass 91, Sixth Conf. on Computer Assurance Systems Integrity, Software Safety, and Process Security, June 25-27, Gaithersburg, Md. Cosponsors: IEEE Aerospace and Electronics Soc., IEEE National Capital Area Council. Contact Dolores R. Wallace, National Inst. of Standards and Technology, Gaithersburg, MD 20899, phone (301) 975-3340, e-mail wallace@swe.ncsl.nist.gov.

First Int'l Conf. on Artificial Intelligence in Design, June 25-27, Edinburgh, Scotland. Contact Helen Hodge or Tom Whiting, Butterworth Scientific, Westbury House, Bury Street, Guildford, Surrey, GU2 5BH, UK, phone (0483) 300-966, fax (0483) 301-563.

Arith 10, 10th Symp. on Computer Arithmetic, June 26-28, Grenoble, France. Cosponsors: ACM et al. Contact Jean-Michel Muller, Lab. LIP-IMAC, Ens. Lyon, 69364 Lyon Cedex 07, France, phone 33 (72) 72-8229.

Third Int'l Conf. on Software Eng. and Knowledge Eng., June 27-29, Skokie, Ill. Sponsors: Knowledge Systems Inst. et al. Contact W.D. Hurley, Computer Science Dept., Alumni Hall, Univ. of Pittsburgh, Pittsburgh, PA 15260, phone (412) 624-8843, e-mail hurley@cs.pitt.edu.

July 1991

CAR 91, Fifth Int'l Symp. on Computer-Assisted Radiology, July 3-6, Berlin. Sponsor: Tech. Univ. Berlin. Contact Heinz U. Lemke, Inst. for Tech. Computer Science, Sekr CG-FR3-3, Franklinstrasse 28-29, D-1000, Berlin 10, Germany, phone 49 (30) 314-73100; or Michael H. Rhodes, Toshiba America MRI, 280 Utah Ave., South San Francisco, CA 94080, phone (415) 875-2909.

SIGGraph 91, July 30-Aug. 1, Las Vegas. Sponsor: ACM. Contact Assoc. for Computing Machinery, 11 W. 42nd St., New York, NY 10036, phone (212) 869-7440.

August 1991

SSD 91, Second Symp. on Large Spatial Databases, Aug. 28-30, Zurich, Switzerland. Contact H.J. Schek, Inst. für Informations Systeme, Eth Zentrum, 8092 Zurich, Switzerland, phone 41 (1) 254-7240.

September 1991

VLDB 91, 17th Int'l Conf. on Very Large Data Bases, Sept. 3-6, Barcelona, Spain. Sponsors: IEEE Computer Soc. Tech. Committee on Data Eng. et al. Contact Guy M. Lohman, IBM Almaden Research Center, Dept. K55, Bldg. 801, 650 Harry Rd., San Jose, CA 95120-6099, e-mail lohman@ibm.com (Internet), lohman@almaden (Bitnet).

Compsac 91, 15th Int'l Computer Software and Applications Conf., Sept. 11-13, Tokyo. Cosponsor: Information Processing Soc. of Japan. Contact Stephen S. Yau, Univ. of Florida, CIS Dept., Rm. 301, Gainesville, FL 32611, phone (904) 335-8006.

October 1991

IEEE Workshop on Visual Motion, Oct. 6-9, Princeton, N.J. Contact Thomas S. Huang, Coordinated Science Lab, Univ. of Illinois, 1101 W. Springfield Ave., Urbana, IL 61801, phone (217) 333-6912.

CSEE 91, Fifth SEI Conf. on Software Eng. Education, Oct. 7-8, Pittsburgh. Contact James E. Tomayko, Software Engineering Inst., Carnegie Mellon Univ., 4500 Fifth Ave., Pittsburgh, PA 15213-3890, phone (412) 268-6806, fax (412) 268-5758.

Workshop on Experimental Distributed Systems, Oct. 12, Huntsville, Ala. Contact Raif M. Yanney, TRW, 1 Space Park, DH2/2328, Redondo Beach, CA 90278, phone (213) 764-6033.

11th Symp. on Mass Storage Systems, Oct. 13-17, Monterey, Calif. Contact Bernard T. O'Lear, NCAR, PO Box 3000, Boulder, CO 80307, phone (303) 497-1268.

RIDT 91, Second Int'l Workshop on Raster Imaging and Digital Typography, Oct. 14-15, Boston. Contact Robert A. Morris, Mathematics and Computer Science Dept., Univ. of Massachusetts at Boston, Harbor Campus, Boston, MA 02125-3393, phone (617) 287-6466, e-mail ridt91-request@cs.umb.edu.

ICCD 91, IEEE Int'l Conf. Symp. on Computer Design, Oct. 14-16, Cambridge, Mass. Cosponsors: IEEE Computer Soc. and IEEE Circuits and Systems Soc. Contact ICCD 91, IEEE Computer Soc., 1730 Massachusetts Ave. NW, Washington, DC 20036-1903, phone (202) 371-1013.

Sixth Int'l Workshop on Software Specification and Design, Oct. 25-26, Como, Italy. Contact Jean-Pierre Finance, CRIN, Campus Scientifique, BP 239 54000 Nancy, France, e-mail finance@loria.crin.fr.

ITC 91, Int'l Test Conf., Oct. 26-30, Nashville, Tenn. Cosponsor: IEEE Philadelphia Section. Contact IEEE Computer Soc., 1730 Massachusetts Ave. NW, Washington, DC 20036-1903, phone (202) 371-1013.

November 1991

ICCAD 91, IEEE Int'l Conf. on Computer-Aided Design, Nov. 11-14, Santa Clara, Calif. Cosponsor: IEEE Circuits and Systems Soc. Contact ICCAD 91, IEEE Computer Soc., 1730 Massachusetts Ave. NW, Washington, DC 20036-1903, phone (202) 371-1013.

Supercomputing 91, Nov. 18-22, Albuquerque, N.M. Cosponsor: ACM. Contact Raymond L. Elliott, Computing and Comm. Div., MS B260, Los Alamos Nat'l Lab, Los Alamos, NM 97545; or Supercomputing 91, IEEE Computer Soc., 1730 Massachusetts Ave. NW, Washington, DC 20036-1903, phone (202) 371-1013.

BOOLE

Version 1.4

The software tool of choice for development and delivery of approximate reasoning models.

Arbitrary boolean formulas and a complete set of coefficients of relation allow precise descriptions of uncertainty.

Solver based on the Binomial Polytopes algorithms for unmatched performance and reliability.

Requires hardware capable of running PC/MS-DOS.

Manual, software: $385.00
Visa MC Check

URSIC Computing

5210 Trafalger Place
Madison, WI 53714
(608) 241-0651
Mon. - Fri. 9am - 4pm

Reader Service Number 8

CAREER OPPORTUNITIES

RATES: $12.00 per line, $120 minimum charge (up to ten lines). Average five typeset words per line, eight lines per column inch. Add $10 for box number. Send copy at least one month prior to publication to: **Heidi Rex or Marian Tibayan, Classified Advertising, COMPUTER Magazine, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720; (714) 821-8380.**

SUNY STONY BROOK
The Department of Electrical Engineering

SUNY Stony Brook invites applications for anticipated openings for all professor ranks. These positions will be tenure track with salary negotiable. The Department has undergraduate and graduate programs and extensive research activities. The areas of prime interest are computer engineering, digital systems, machine vision, VLSI design, and robotics. The Department has extensive computing facilities. Applicants in other areas will be considered. The Department has close relations with high technology industry. Stony Brook combines the attraction of semi-rural location with proximity to the resources of the New York City area. Please submit resume to: Prof. Kenneth L. Short, Chairman, Department of Electrical Engineering, SUNY Stony Brook, Stony Brook, NY 11794-2350. SUNY Stony Brook is an affirmative action/equal opportunity educator and employer. AK 175.

UNIVERSITY OF PITTSBURGH
Department of Computer Science

The Department of Computer Science invites applications for one or more tenure-track faculty positions at the assistant professor level areas of interest are Systems, Data Base, and Artificial Intelligence. The starting date will be September 1, 1991. Responsibilities include research, supervision of graduate student research (Ph.D. and M.S.), and graduate and undergraduate teaching. Candidates should have a Ph.D. in computer science and a strong interest in both research and teaching.

The Department currently has twenty-three full-time faculty members and supports strong graduate and undergraduate programs. Departmental resources include an excellent research library and extensive computing facilities including a network of SUN and Xerox 1100-series (Dandelion) workstations, a VAX 11/780 (under BSD UNIX), an Intel iPSC/2 hypercube, a variety of micro-computers, and several graphics systems. The research systems are accessible via the Department's Ethernet-compatible LAN. Convenient access is also provided to the extensive general computer facilities of the University as well as to other networks (e.g., ARPANET, CSNET). The Department operates the *Center for Parallel, Distributed and Intelligent Systems* (CPDIS) to provide an environment for innovative research in computer science. Since the University of Pittsburgh is a founding member of the Pittsburgh Supercomputing Center and an affiliate member of the Software Engineering Institute, the Department of Computer Science has access to the Cray X-MP/48 of PSC and the software engineering expertise at SEI.

Please send your resume to: Dr. Rami Melhem, Chair of Faculty Search, Department of Computer Science, University of Pittsburgh, Pittsburgh, PA 15260.

Pitt is an equal opportunity/affirmative action employer and especially encourages women and members of ethnic minorities to apply.

STANFORD UNIVERSITY
Faculty Openings

The Departments of Computer Science and Electrical Engineering have openings for both tenure-track and research faculty positions. These positions are affiliated with the Computer Systems Laboratory, a joint laboratory within the two departments focusing on teaching and research in computer systems.

Applications are invited. Applicants should have received a Ph.D. in Computer Science or Electrical Engineering prior to September, 1991. We are looking for candidates with demonstrated research ability in one or more areas of computer hardware and software, including programming languages, and operating systems. All applicants will be considered for positions beginning in the Fall Quarter of 1991.

Stanford University is an affirmative action, Equal Opportunity Employer and welcomes applications from women and minority groups. Please submit a detailed resume including FIVE references, with at least one reference outside your current institution. Applications should be sent to: Professor John Hennessy, Search Committee Chairman, Computer Systems Laboratory, Department of Electrical Engineering, Stanford University, Stanford, California 94305-4055. Applications should be sent prior to February 1, 1991.

THE AMERICAN UNIVERSITY IN CAIRO
Faculty Position

One Assistant, Associate, or Full Professor needed to teach, in English, undergraduate courses in Computer Science including theory of computing and design of algorithms. Ph.D. required. Rank, salary based on qualifications and experience. Two-year appointment (renewable) begins September 1991. For expatriates, housing, roundtrip air travel, plus schooling for two children included. Write with curriculum vitae to: Dean George H. Gibson, The American University in Cairo, 866 United Nations Plaza, Suite 517, New York, New York 10017, preferably before December 15.

BOISE STATE UNIVERSITY
and
HEWLETT-PACKARD

Boise State University, in cooperation with Hewlett-Packard, invites applicants for a tenure-track position in the Department of Mathematics for the fall of 1991. Applicants should have a Ph.D. in computer science or a related field with preference given to candidates with expertise in software engineering, artificial intelligence, or database systems. Industry experience is beneficial. Responsibilities will include teaching, research, and service split between the University and Hewlett-Packard's Boise site. Salary will be commensurate with rank and qualifications.

Boise State University is located in Boise, Idaho which is the state's largest city and is the site of many technology-based corporations. In the September issue of *Money* magazine, Boise was ranked 37th out of 300 of the best U.S. cities in which to live. The city is the political, economic, and cultural hub of Idaho. The person selected for this position can expect to find world-class outdoor recreational opportunities, an extraordinary quality of life, and a highly collegial faculty with balanced teaching and research interests.

A letter of application, a vita, three letters of reference, and graduate transcripts are required. Contact Dr. Phillip Eastman, Computer Science Search Committee Chair, College of Arts and Sciences, Boise State University, Boise, Idaho 83725. Screening will begin on January 15, 1991. BSU & HP are EEO/AA employers.

FLORIDA STATE UNIVERSITY
Department of Computer Science

The Department of Computer Science is seeking qualified candidates for expected tenure track positions. Areas of special interest include, but are not limited to, software engineering, operating system, networking, programming languages, computer graphics, database, and computer architecture.

Our department is a young and rapidly growing one. It offers B.S., M.S., and Ph.D. degrees, and it places a very strong emphasis on excellence in research and teaching.

Research facilities include a CRAY Y-MP/432, a CM2 Connection Machine, an ETA-10, Internet connections through SURANET and ESnet, a group SUN 3 and SUN 4 workstations and servers, a VAX 11/780, and a Symbolics 3640.

Applicants should have a Ph.D. in computer science or a closely related area, and a strong commitment to teaching and research.

Send a resume and names of at least three professional references to:

Dr. Abe Kandel, Chairman
Department of Computer Science B-173
Florida State University
Tallahassee, FL 32306-4019

Florida State University is an Equal Opportunity Affirmative Action Employer.

UNIVERSITY OF CALIFORNIA AT IRVINE
Faculty Positions in Computer Science

The Department of Information and Computer Science (ICS) is actively recruiting faculty AT ALL LEVELS. We have dynamic research groups in the areas of computer systems design, parallel processing, artificial intelligence, computer networks and distributed processing, software, social and managerial analysis of computing, and theory. We are continuing to build on these areas of strength. We are also interested in developing new strength in computational biology, integrated systems, computer-supported cooperative work, databases, design tools, and model-based reasoning. We will sympathetically review applications from very strong candidates in all areas of computer science.

We are looking for new faculty with strong research records who would thrive in a highly productive but friendly setting, and who would like to join us in exploring the nature of computing, broadly defined. We specially encourage application from exceptionally distinguished candidates for senior positions.

The ICS Department is an independent academic unit reporting to the Executive Vice Chancellor. ICS faculty emphasize core computer science as well as research in emerging areas of the discipline, with effective interdisciplinary ties to colleagues in neurobiology, cognitive science, management, engineering, and the social sciences. The department currently has 30 full-time faculty positions and over 120 Ph.D. students, with major support from the administration to expand and to strengthen the research environment.

Annual research funding from contracts and grants from agencies such as DARPA, NSF, and ONR, currently total over $6.5 million. In 1986 the software group was awarded a Coordinated Experimental Research (CER) grant from the National Science Foundation. This support has fostered the creation of a Research Laboratory for software engineering, in which major studies of the development and evaluation of software technology are undertaken. High quality research has also fostered the creation of a Research Laboratory for computer systems design that deals with methodologies and tools for the design of complex computational systems. A third Research Laboratory, in Artificial Intelligence, is planned.

Department equipment includes approximately 175 workstations, primarily Sun-3's and Sun-4s. Two large multiprocessor Sequents and a Hypercube are available, as well as approximately 300 Macintosh Plus's and II's. All our major workstations and computers are tied together with networks, which are gatewayed to the campus network, and from there, to regional, national, and international networks (Darpa Internet, CSnet, Bitnet, etc.). In addition, department members have access to campus-wide computing resources as well as regional supercomputer access.

UC-Irvine is located in Orange County, three miles from the Pacific Ocean near Newport Beach, and approximately halfway between Los Angeles and San Diego. The campus is situated in the heart of a national center of high-technology enterprise. It is growing rapidly and offers exciting professional and cultural opportunities. Salaries and benefits are competitive. Special housing assistance is available from the university, including newly built, for-sale housing within short walking distance from the Department.

Send resumes and names of four references to:

Chair, Faculty Recruiting Committee
Department of Information and Computer Science
University of California-Irvine
Irvine, CA 92717

The ICS Department has several vacant positions and application screening will begin immediately upon receipt of curriculum vitae. Maximum consideration will be given to applications received by January 31, 1991.

The University of California is an Affirmative Action/Equal Opportunity Employer. The Department of ICS is particularly interested in receiving applications from women and minority candidates.

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
The Department of Computer Science

The Department of Computer Science, University of Illinois at Urbana-Champaign, anticipates possible tenure or tenure-track appointments in several disciplines. Applicants must have outstanding academic credentials and an ability to teach effectively at both the graduate and undergraduate levels. Selected candidates will be expected to initiate and carry out independent research and to perform academic duties associated with our BS, MS, and PhD programs. Since the department is large, several vacancies are usually filled each year. A continuing search is conducted throughout the year to fill open positions. All applications received by December 1, 1990, will receive equal consideration, although a search will continue until a position has been filled. PhD or imminent completion required. Salary and rank open, based on qualifications. Starting dates are January 6, 1991 or August 21, 1991, although other dates are negotiable. Send resume including names of three references to: Duncan H. Lawrie, Head, Department of Computer Science, 1304 W. Springfield Avenue, Urbana, IL 61801. Phone (217) 333-6454. The University of Illinois is an affirmative action, equal opportunity employer.

CARNEGIE MELLON UNIVERSITY
Faculty Positions in Computer-Aided Engineering and Management

Carnegie Mellon University, Department of Civil Engineering, invites applications for tenure-track positions at the rank of Assistant Professor or above in (1) Computer-Aided Engineering (CAE) and (2) Planning, Management and Automation (PMA), starting January or September, 1991. A background in engineering and a Ph D. degree are required. CAE applicants should have an MS degree in Computer Science or equivalent; PMA applicants should have a strong background in at least one interest area listed.

Within CAE, areas of interest include artificial intelligence, knowledge-based systems, database management, parallel computations, design methodology, graphics, user-interfaces, visualization, geometric modeling and large-scale systems design. Within PMA, areas of interest include construction project management, construction automation, environmental management, real time control, remote sensing, and transportation systems.

Responsibilities include undergraduate and graduate teaching, advising, developing curriculum, supervising graduate students, and establishing a funded research program.

The department offers excellent opportunities for professional and intellectual growth and maintains close ties with the School of Computer Science and the Engineering Design Research Center.

Applicants should send a curriculum vita and names of at least three references to: D.R. Rehak, Department of Civil Engineering, Carnegie Mellon University, Pittsburgh, PA 15213. INTERNET: REHAK@CE.CMU.EDU. FAX: (412)268-7813.

Closing date for applications is 1/1/91, or until the positions are filled. AA/EOE.

FACULTY FOR EUROPE AND ASIA

Planning a sabbatical or leave of absence? The University of Maryland University College seeks excellent lecturers for undergraduate computer science, computer applications, and information systems management courses on U.S. military bases in Europe and Asia and in the Pacific. Renewable annual appointments begin August 1991. Minimum requirements include a master's degree in computer science or a related field, recent college teaching experience, and U.S. citizenship. Benefits include transportation and military base privileges. Frequent travel and the cost of schooling make these positions difficult for those with children. Send resume to Dr. Ralph E. Millis, Overseas Programs, The University of Maryland University College, College Park, MD 20742-1642. AA/EEO.

CONCORDIA UNIVERSITY
Department of Computer Science

Applications are invited for Visiting Faculty, Post-Doctoral and Research Associate positions in the following areas: pattern recognition and machine intelligence, expert systems, databases, programming languages, software engineering, VLSI architecture, numerical analysis. Applicants should have a Ph.D. degree in computer science or related field with a good research record. The Department currently has 26 full-time professors. It offers both undergraduate and graduate programs up to the Ph.D. level with an enrollment of over 1000 students. The language of instruction is English. The Department and the University have excellent research and teaching facilities, and support staff.

Apply giving resume and the names of at least three references to:

Dr. T.D. Bui, Chair
Department of Computer Science
Concordia University
1455 de Maisonneuve Blvd. West
Montreal, Quebec H3G 1M8 Canada

In accordance with Canadian Immigration requirements, priority will be given to Canadian citizens and permanent residents of Canada.

CLEMSON UNIVERSITY
Department of Electrical and Computer Engineering

The Department of Electrical and Computer Engineering at Clemson University invites applications for tenure/tenure-track positions, primarily at the assistant professor level. A Ph.D. is required. Candidates with research interests in one of the following areas are sought: communications and digital signal processing; networks and distributed computing; quantitative analysis of computer architectures; electromagnetic analysis of active devices (microwave, millimeter wave, or electro-optic); and multidisciplinary applications in power systems (e.g. power and artificial intelligence; power and computer communications). A successful candidate must exhibit exceptional potential for research and teaching.

Clemson University's College of Engineering was listed as one of the United States' "up-and-coming" engineering graduate programs in the March 19, 1990, issue of *U.S. News and World Report*. The ECE Department comprises thirty-six full-time faculty, approximately 700 undergraduate students, and 140 graduate students. It offers B.S., M.S., and Ph.D. degrees in both electrical engineering and computer engineering. Facilities and/or groups bearing on the areas indicated above include the Clemson University Electric Power Research Association (CUEPRA); a microcircuits reliability research group with a class 100 clean room; automated microwave measurement facilities to 26 GHz; an image processing laboratory; a Center for Computer Communications Systems.

Resumes, supported with a list of references, should be sent to L. Wilson Pearson, Head; Department of Electrical and Computer Engineering; 102 Riggs Hall; Clemson University; Clemson, SC 29634-0915. Initial screening of applicants will begin January 15, 1991 and continue until positions are filled. Clemson University is an Equal Opportunity/Affirmative Action Employer.

UNIVERSITY OF SOUTH FLORIDA
Department of Computer Science & Engineering

The Department of Computer Science & Engineering is seeking applications for four tenure track faculty positions, three at our Tampa Campus and one at the Sarasota Campus, starting in the 1991-92 academic year. The Department offers Bachelor's degrees in Computer Science, Computer Engineering, and Information Systems, and has Master's and Ph.D. programs. The present faculty size is 15.

The Department shares a new 12 million dollar building with the Department of Electrical Engineering. The Department research network includes a substantial number of SUNs, an INTEL 2/386 hypercube, and a variety of specialized graphics and image processing equipment. Additional IBM, HP, and DEC systems are currently being added to support the enhanced Information Systems program. Also, computing resources are available on the College computing network and the University network. The departmental faculty are actively engaged in research activities supported by Federal, State and industrial sources.

The University of South Florida is the second largest university in Florida, with an enrollment of over 30,000. It is located on a 1,700 acre campus in the city of Tampa, one of the largest and fastest growing metropolitan areas in Florida, offering a wide variety of cultural, entertainment, sports and outdoor activities. The quality of life is excellent and the cost of living is moderate. The area near the University is experiencing dramatic growth in high-technology industry and medical facilities. The faculty of the Department have interactions with a number of the companies in the area including AT&T, E-Systems, GTE Data Services, Harris, Hercules, Honeywell, and IBM. A PBS public-TV channel is located on campus; other television links tie the University to industry and remote campuses.

Applicants should send a resume and arrange to have three letters of reference sent to Faculty Search Committee, Computer Science & Engineering, University of South Florida, Tampa, FL 33620.

The University of South Florida is an equal opportunity employer.

UNIVERSITY OF ARKANSAS
J. William Fulbright College of Arts and Sciences

The Department of Computer Science welcomes applications for a tenure track assistant professorship beginning in August, 1991. The Department offers BA, BS and MS degrees and has a Ph.D. program pending implementation.

A candidate must have a Ph.D. or equivalent degree in computer science and be strongly committed to research and teaching. The field of research is open, but a candidate must teach programming languages at the undergraduate and graduate levels and software engineering at the undergraduate level. Teaching other undergraduate ACM curriculum courses is also expected.

Applicants should submit a resume and names of three references to: Greg Starling, Department of Computer Science, 232 Science-Engineering Building, University of Arkansas, Fayetteville, AR 72701, Telephone: (501) 575-6427. Email: Starling@uafsysb.uark.edu.

The University is an EO/AA Employer.

SOFTWARE APPLICATION DEVELOPMENT CONSULTANT

Software Application Development Consultant for proprietary software marketing, support & consulting company. Job includes on-site client consulting in development of business computer based information systems, conferring with technical & management staffs of clients for development of business applications & effective use of database & 4GL technologies for on-line & Batch computer systems, using NOMAD language on IBM SQL/DS based DBMS operating in VM & MVS operating systems on IBM 370 mainframe units, using REXX, EXEC, Cobol and JCL languages, use of relational database management & database administration principles & theories. Projects of varying size and complexity requiring integration with IBM 370 VM, MVS, DOS/VSE operating system & DEC VAX minicomputers for systems development. In excess of 70% travel to client sites across U.S. Requires a Bachelor's Degree in Business and two years' experience as a Business Systems Analyst/Programmer, experience to include at least one year of application development using the NOMAD language. $865 per week. 40 hours per week. Apply at the Texas Employment Commission, Houston, Texas, or send resumes to the Texas Employment Commission, Tec Building, Austin, Texas 78778. J.O. #5515163. Ad Paid By An Equal Employment Opportunity Employer.

UNIVERSITY OF SOUTHERN CALIFORNIA

The Computer Engineering Division of the Department of Electrical Engineering-Systems at the University of Southern California is expanding, and looking to fill positions at the Assistant, Associate, and Full Professor level in the following areas: VLSI/CAD, networks (optical type), and architecture with an emphasis on hardware. Additionally, we are looking for a full-time instructor (M.A. only required) to support the Computer Science/Computer Engineering undergraduate degree program. For all openings, please send a resume and the names of at least three academic references to Jerry M. Mendel, Chairman, Department of Electrical Engineering-Systems, University of Southern California, Los Angeles, CA 90089-0781. USC is an equal opportunity/affirmative action employer.

COMPUTER CONSULTANT/ LEAD SYSTEMS ENGINEER

Computer Consultant/Lead Systems Engineer needed to analyze, design, develop, and implement complex computer applications to meet client's needs. Provide project leadership for Systems Development Projects. Estimate time, effort and cost for Systems Development Projects. Provide technical support to Marketing Responsibility Projects. Conduct Operational Systems Audit for clients. Prepare written summary reports to client and management. Requires a Master's degree in Computer Science or Business Administration, with specialization in Systems Management and four years experience in job offered or four years directly related Systems Analyst experience. Requires at least one year experience in the following: (a) Multiple Vendor platforms such as IBM As/400, HP 3000, DEC VAX Series; (b) Databases, such as IBM DB2; and (c) Application knowledge in one of the following: Manufacturing (MRP, Inventory Control, etc.); Banking; Insurance; and Oil and Gas. 40 hour work week. $45,000 per year. Apply at the Texas Employment Commission, Dallas, Texas, or send resume to the Texas Employment Commission, TEC Building, Austin, Texas 78778. Job Order #5424768. Ad Paid By An Equal Employment Opportunity Employer.

STATE UNIVERSITY OF NEW YORK AT BINGHAMTON
Department of Computer Science The Watson School of Engineering

The State University of New York at Binghamton invites applications for tenure-track positions in the Department of Computer Science beginning August 1991. Positions are available at junior and senior levels and the salary is competitive. Preferred areas of specialization include distributed/parallel systems, databases, software engineering, and artificial intelligence, but applicants in all areas of Computer Science will be considered. Applicants must have a Ph.D. in Computer Science or a related area, and possess a strong commitment to research and teaching.

The Department has established Ph.D. and M.S. programs, and an accredited B.S. program. High technology computer-oriented companies in the local area such as IBM, G.E., Link Flight Simulation, and Universal Instruments provide opportunities for industrial collaboration.

Send nominations or applications including a resume and the name of three references to Professor Sudhir Aggarwal, Chairman, Department of Computer Science, The Watson School, State University of New York at Binghamton, P.O. Box 6000, Binghamton, New York 13902-6000. Applications received by January 15, 1991 will receive first consideration.

The State University of New York at Binghamton is strongly committed to affirmative action. We offer access to services and recruit students and employees without regard to race, color, sex, religion, age, disability, marital status, sexual orientation or national origin.

COLUMBIA UNIVERSITY
Department of Computer Science

We are looking for several exceptional people to join our faculty. Tenure-track positions are available at all ranks in all areas.

Our department of nineteen tenure-track and three teaching faculty emphasizes research, and attracts excellent Ph.D. students, virtually all of whom are fully supported. Departmental facilities include many advanced workstations, high-performance servers, and graphic systems, plus state-of-the-art systems designed and built at Columbia for vision, robotics, parallel computation, networking and distributed computing. We are within an hour's drive of the research laboratories of IBM, AT&T, Bellcore, Siemens, Philips, NYNEX, and other leading industrial companies.

Columbia University is one of the oldest and most prestigious universities in the United States, and New York City is one of the cultural, financial, and communications capitals of the world. The department is housed in its own building, and in 1992 we will acquire additional space and facilities in the interdisciplinary Center for Engineering and Physical Science Research now under construction. University-subsidized housing and parking is readily available.

Candidates for assistant professor should exhibit exceptional research promise, while those seeking a more senior position should have an outstanding record of research achievement. Interest and ability in teaching undergraduates and graduates is necessary. Send resume and the names of at least three references to: Prof. John R. Kender, Faculty Search Chairperson, Department of Computer Science, Columbia University, New York, New York 10027.

Columbia University is an Equal Opportunity/Affirmative Action Employer. We encourage applications from women and minorities.

TRANSYLVANIA UNIVERSITY
Computer Science Assistant/Associate Professor

Full-time, tenure track, Ph.D. in computer science or master's in computer science with Ph.D. in a closely related field. Rank and salary dependent upon background. Exceptionally well qualified candidates may be considered for a Bingham Award for Excellence in Teaching; small, "start-up" grants are available for less experienced faculty. This recognition provides a supplement of up to 50% of base salary for the position. Transylvania is a private, liberal arts college with a strong commitment to excellence in undergraduate education. The program in computer science is of long standing and is recognized for its outstanding quality. Please send letter of application, curriculum vitae, and names of three references to Dr. Dwight W. Carpenter, Computer Science Program, Transylvania University, Lexington, KY 40508. An Equal Opportunity Employer.

WRIGHT STATE UNIVERSITY
Department of Computer Science and Engineering

Applications are invited for the position of INSTRUCTOR in Computer Science and Engineering. Successful candidates must have an M.S. degree in Computer Science, Computer Engineering, or related field. Teaching experience strongly preferred as duties will include teaching introductory undergraduate courses in computer science/engineering, preparing classroom materials, grading, and advising. Instructor positions are non-tenure track, nine-month appointments with the likelihood for reappointment. Summer support is usually available for qualified instructors. There are opportunities for pursuing a Ph.D. Wright State University's Computer Science and Engineering department has a variety of dynamic research and education programs. Its laboratory facilities boast a VAX 5810, 30 SUN and DEC workstations networked into extensive University facilities, and Internet access. Wright State University is a state supported university located in the Dayton metropolitan area, a high technology environment composed of several industrial, corporate and military research and development facilities. Review for positions will begin December 1, 1990 and continue monthly until filled. Send a letter of application, resume, names of three references, and official copies of transcripts to: CSNET ADDRESS—amcaulay@cs.wright.edu or Dr. Alastair D. McAulay, Chair, Department of Computer Science and Engineering, Wright State University, Dayton, OH 45435.

An equal opportunity/affirmative action employer.

UNIVERSITY OF WISCONSIN-MADISON
Faculty Position

The Department of Electrical and Computer Engineering invites applications for a possible tenure or tenure-track position. A Ph.D. degree is required, and successful candidates are expected to participate in both teaching and research activities. Applicants in all areas of computer engineering are invited to apply, but the following areas are of special interest: computer architecture, computer networks, VLSI and computer-aided design, microprocessor and minicomputer applications, real-time control and instrumentation applications, and engineering applications of artificial intelligence. Rank and salary will be commensurate with qualifications and experience. Send resume and names of three references to Bahaa E.A. Saleh, Chairman, Department of Electrical and Computer Engineering, University of Wisconsin-Madison, 1415 Johnson Drive, Madison, WI 53706, an equal opportunity/affirmative action employer.

RSA DATA SECURITY INC.

RSA Data Security Inc. invites applications for a position in research and advanced development. RSA Data Security is a small company specializing in the integration of cryptographic technology into commercial systems. Applicants should have a Ph.D. in Computer Science, Mathematics, or a closely related field.

Responsibilities of the position include developing new cryptographic protocols and algorithms; evaluating cryptographic systems; and maintaining an active research program.

Applicants should send a resume and the names of three references to:

RSA Data Security
10 Twin Dolphin Drive
Redwood City, CA 94065
Telephone: (415) 595-8782
Electronic mail: burt@rsa.com

An Equal Opportunity/Affirmative Action Employer.

UTAH STATE UNIVERSITY

Applications are invited for all faculty levels in Computer Science. Qualifications include a doctorate in Computer Science or a closely related field; a strong commitment to teaching at both the undergraduate and graduate levels; and a similar commitment to research. Specializations of particular interest include Parallelism, Software Engineering, and Artificial Intelligence.

The department currently offers B.S. and M.S. degrees in Computer Science. Successful candidates would be expected to assist in the development of a Ph.D.

The university is located in a beautiful mountain valley with easy access to recreational and cultural activities.

Send resumes and names of 3 references to: Gregory Jones, Computer Science, Utah State University, Logan, Utah, 84322-4205. Positions will open January and remain open until filled. U.S.U. is an EO/AAE employer.

SOFTWARE DEVELOPMENT ENGINEER

Software Development Engineer needed to design, develop, implement and test NEAX 61E System design features using Assembler and PL-C languages to meet LSSUR requirements for user interface and peripheral device assess design. Conduct logical analysis of scientific, engineering and other technical problems and formulate mathematical models of problems for solution by computer. Debug and problem report resolution. Analyze, review and develop software amendments, and maintenance in UNIX for software quality control. Diagnose, define and resolve software problems and provide hardware performance tuning and maintenance service. Plan and prepare technical reports. Requires a Bachelor's degree in Engineering or related area and two years experience in job offered or two years related software and hardware engineering or will consider a combination of additional academic coursework and experience totalling two years experience in lieu of Bachelor's and two years. Requires at least one year experience or coursework in both high level languages such as Pascal and C as well as Assembler language. 40 hour work week. $37,600.00 per year. Apply at the Texas Employment Commission, Dallas, Texas, or send resume to the Texas Employment Commission, TEC Building, Austin, Texas 78778. Job Order #5426688. Ad paid for by an Equal Employment Opportunity Employer.

NORTHWESTERN UNIVERSITY
The Robert R. McCormick School of Engineering and Applied Science
The Department of Electrical Engineering and Computer Science

Tenure track faculty positions available at all levels.

The department's objective in recruiting new faculty is to build a world-class research program in computer science and engineering. Our goal is to complement the strong success of the Institute for the Learning Sciences at Northwestern University under the direction of Roger Schank. We are interested in exceptional applicants with expertise in all aspects of: artificial intelligence, system software design, architecture, theoretical computer science, and related areas.

Northwestern University is an equal opportunity, affirmative action educator and employer. Applications from women and minorities are especially encouraged. Employment verification required upon hire.

Applicants should have attained a doctoral degree in computer science, computer engineering, electrical engineering, or a closely related field by the starting date of the appointment. Candidates for higher level positions must have a demonstrated record of research excellence in industrial or academic settings.

Interested persons should send their resumes and the names and addresses of at least four references to:

Professor A. H. Haddad, Chairman
Department of EECS
Northwestern University
2145 Sheridan Road
Evanston, Illinois 60208-3118

Scientific Programmer

We are seeking a Scientific Programmer to develop intelligent graphical interfaces and to explore the definition of problem-solving methods for expert systems that plan medical therapy. Duties include program design, implementation, and documentation; conducting basic research; supervising students; and preparing technical reports. A Ph.D., or imminent completion, with emphasis in adaptive intelligent interfaces is required. Familiarity with machine learning techniques, knowledge-representation formalisms, expert-system construction, window-based user interface design, and software project management is necessary. Knowledge of C, C++, Lisp, UNIX, X Windows, Motif, LEX and YACC, compiler construction, and relational databases is essential. $50,000/yr. Send vitae before December 1, 1990 to: IEEE Computer Society, P.O. Box 3014, Dept. MT-100, Los Alamitos, CA 90720-1264.

GEORGE MASON UNIVERSITY
Faculty Positions in Computer Science

The Department of Computer Science at George Mason University is seeking applicants for tenure-track and tenured faculty positions at the ranks of assistant, associate, and full professor. We are especially interested in persons whose specialty areas are algorithm analysis, automata and formal language theory, compilers, computer networks, and operating systems. We are primarily interested in full-time faculty, but may consider visiting faculty as well. The nominal date of appointments is September 1991.

George Mason University is located in Fairfax County, Virginia, just fifteen miles from the exciting scientific, cultural, and tourist attractions of Washington, D.C. The Department of Computer Science is in the School of Information Technology and Engineering, which has made a commitment to establish and maintain the outstanding academic and research record in the Department. There are also numerous opportunities for government and industrial interaction in this suburban high technology area.

For full consideration please send a letter of application, a detailed resume, samples of recent publications, and the names of four references and other supporting material to Professor Alan M. Davis, Chair, Recruitment Committee, Department of Computer Science, George Mason University, Fairfax, VA 22030-4444, or call at (703) 323-2713 (e-mail: adavis@gmuvax.gmu.edu). Applications are sought by February 1, 1991, and these will receive first priority. AA/EOE.

UNIVERSITY OF HOUSTON
The Department of Electrical Engineering

The Department of Electrical Engineering invites applications for tenure-track positions in the areas of applied electromagnetics, biomedical engineering, communications, computer engineering, electronics and microelectronics. Openings are available at all ranks. Please send resumes to Dr. J.C. Wolfe, Department of Electrical Engineering, University of Houston, Houston, TX 77204-4793. An Equal Opportunity Employer.

UNIVERSITY OF WASHINGTON
Department of Computer Science and Engineering

The Department of Computer Science and Engineering at the University of Washington expects to have one or more tenure-track openings starting in the 1991-92 academic year. We seek outstanding applicants who add to our existing research strengths, particularly in programming languages, compilers, graphics, hardware/computer engineering, and software engineering, or who bring significant new research strength to our department.

A moderate teaching load allows time for quality research and close involvement with students. We expect applicants to have a strong commitment to both research and teaching, and an outstanding record of research for their level.

The department may also have visiting positions that would require both teaching and research. It may be possible to hold these for portions of the 1991-92 academic year.

Interested applicants should send a letter of application, a resume, and the names of four references to Faculty Recruiting Committee, Department of Computer Science and Engineering FR-35, University of Washington, Seattle, Washington 98195. Candidates are encouraged to apply as early as possible.

The University of Washington is an Affirmative Action/Equal Opportunity Employer. The Ph.D. is required for these positions.

UNIVERSITY OF SOUTHERN CALIFORNIA
Computer Science

Applications are invited for senior or junior tenure-track faculty positions in Computer Science. Candidates should have a strong commitment to research. Openings exist in most areas of experimental and theoretical computer science; with emphasis on operating systems, networks, compilers, artificial intelligence, graphics, geometric modeling, software engineering, and parallel algorithms and languages. The department currently has 19 full-time faculty, 135 doctoral students and annual research support excess of $3 million.

Research computing facilities include some 50 Sun 3 and Sun 4 workstations, a 32-node Intel IPSC/2-d5 hypercube, 4 TI Explorer Lisp machines, and numerous personal computers. An 8-processor Alliant and a 64K processor Connection Machine (located at the USC Information Sciences Institute) are also available for research via high-bandwidth networks. Research laboratories are dedicated to Brain Modeling, Artificial Intelligence, Programmable Automation, Robotics, Databases, Networking, Graphics and Animation, and Software Engineering. Teaching is supported by an additional 100 Sun workstations. Interested candidates should send a resume and a list of references to: Prof. Ellis Horowitz, Acting Chairman, Computer Science Department, SAL 200, University of Southern California, Los Angeles, CA 90089-0782.

USC is an Equal Opportunity/Affirmative Action Employer.

UNIVERSITY OF UTAH
Department of Computer Science

Applications are invited for tenure-track position in Computer Science, available in September 1991. One position is in the area of Systems and the other in Robotics/Computer Vision. All applicants should have a Ph.D. in Computer Science. A strong commitment to research and teaching is required. Send CV and names of at least three references to: Faculty Recruiting Committee, Department of Computer Science, University of Utah, Salt Lake City, Utah 84112. An Equal Opportunity/Affirmative Action Employer.

GEORGIA INSTITUTE OF TECHNOLOGY
College of Computing

Georgia Tech, in recognition of the centrality of computing to its future excellence, has created a College of Computing to provide academic and intellectual leadership in broad areas related to computer science. The College has a faculty of 37 and a research faculty of 14. Its educational activities include an undergraduate computer science major accredited by CSAB, a Masters program with over 100 students, a Ph.D. program with over 100 students, participation in several interdisciplinary programs, and active programs of continuing education. Its research activities span the core areas of computer science, but also include interdisciplinary programs in cognitive science, computer graphics and visualization, and discrete mathematics. Additional opportunities exist in several research centers that will be a part of the College, including work in software, graphics and visualization, and information systems. The School is in the fourth year of a five year NSF CER grant. This grant supports experimental work in parallel and distributed computing. The college's laboratory facilities consist of networks of high-performance Unix workstations as well as experimental parallel computer equipment. Additional laboratories offer robotics and vision equipment, facilities for research in graphics and visualization, telecommunications hardware and software, LISP machines, and other area-specific equipment.

The College invites applications for faculty positions at all levels and in all areas of computing in which the College is active. Applicants at the Associate Professor and Professor level are especially encouraged. Applicants should have a record of outstanding research accomplishments and a strong commitment to teaching and service. Applicants who expect to receive a Ph.D. degree by Fall 1991 and who show high potential for innovative contributions to the College's mission are also invited.

Georgia Tech is located in Atlanta the site of the 1996 Olympics. The campus will be centrally involved in the preparation for and staging of the games. Atlanta experiences a mild sunbelt climate and is the center of commerce in the Southeast, offering a diverse economy and good employment opportunities in all professional areas. It offers good cultural and recreational oportunities, extremely attractive residential neighborhoods, and affordable housing.

Candidates should send complete resumes and names of at least three references to: Professor Karsten Schwan; Chair, Faculty Search Committee; College of Computing; Georgia Institute of Technology; Atlanta, Georgia 30332-0280; PHONE: (404) 894-2598; e-mail: recruiting@cc.gatech.edu; FAX: (404) 853-9378.

Georgia Tech is an equal opportunity employer.

THE UNIVERSITY OF TAMPA
Computer Information Systems

The College of Business, CIS Department, announces a CIS position with teaching responsibilities in all areas of the curriculum, commencing August 1991. A Ph.D. in CIS/MIS/CS is preferred although ABD's acceptable. Knowledge of Artificial Intelligence, Object Oriented Design, Computer Graphics, and/or Decision Support Systems is desired. The College of Business rewards teaching excellence and encourages faculty contact with the business community. This position is a tenure-track with rank and salary dependent on qualifications.

Applicants should send a letter of application and an attached vita by December 1, 1991 to:

Dr. Gordon Couturier, Coordinator
College of Business, CIS Department
The University of Tampa
401 West Kennedy Boulevard, Box 131F
Tampa, FL 33606-1490

The University of Tampa is an Equal Opportunity, Affirmative Action Employer and encourages women and minorities to apply.

Software Engineers

Schlumberger, the industry leader in high-tech oilfield services, has openings at its Systems Center where we design and develop computer-based systems for geoscience data acquisition, transmission, and interpretation. Because this work is challenging and fast-paced, it requires individuals with exceptional talent, initiative, and self-confidence. Our new campus-style facilities are located in the Hill Country of Austin.

Openings exist for software developers in the following areas:

- Communications
- Databases
- Graphics
- Software engineering
- System software

An advanced degree in Computer Science or Electrical Engineering is preferred along with familiarity with UNIX or VMS platforms. Additional experience with object-oriented programming, X human interfaces, and other state-of-the-art techniques are appreciated. If you meet these requirements and wish to investigate the opportunities available at Schlumberger's Austin System Center, please send your resume to:

Schlumberger Well Services
ATTN: Personnel Manager
P.O. Box 200015
Austin, Texas 78720

Schlumberger

An Equal Opportunity Employer

Research Scientists

Vehicles that are developed on the leading edge of automotive technology for the North American market begin the journey at **Nissan Research & Development, Inc.** Currently, we have opportunities for Research Scientists with proficiency in computational mechanics and structural analysis using Finite Element Methodology.

Two to three year contracts are available for work in our Ann Arbor, MI facility or our Central Engineering Laboratory in Japan. Master of Science degree in a related field is required; PhD is preferred.

If becoming part of a dynamic company on the leading edge of automotive technology interests you, please forward a resume, in confidence, to: **Nissan Research & Development, Inc., P.O. Box 8036-DH, Plymouth, MI 48170-8036.** Equal Opportunity Employer.

NISSAN

UNIVERSITY OF CALIFORNIA, SANTA CRUZ

Computer Engineering Department invites faculty applications for two positions.

One Associate or Full Professor, and one Assistant with an application closing date of January 1, 1991. Preference will be given to candidates in the following areas:

One with research and teaching interests in graphics and imaging, with emphasis on applications and systems. Expertise in graphical interfaces and workstations is important. (Provision #233-901).

One with research and teaching concentration in computer system architecture. It is desired that this person have strong hardware and design interests, and would emphasize system design exploiting our department's strengths in VLSI/CAD. (Provision #175-890).

Salary range for the tenured position, $53,400-$77,600 and for the tenure-track position, $44,200-$46,400 (9 month basis).

A Ph.D. in Computer Engineering, Electrical Engineering, Computer Science or equivalent is required. For complete information please contact: Chair, Computer Engineering Faculty Search Committee, Baskin Center for Computer Engineering & Information Sciences, Applied Sciences Building. University of California, Santa Cruz, CA 95064. (Questions may be sent via email to recruit@saturn.ucsc.edu or recruit@ucscrls.bitnet.) UCSC is an EEO/AA/TRCA employer.

WASHINGTON STATE UNIVERSITY

The Department of Electrical and Computer Engineering at Washington State University invites nominations and applications for the Boeing Endowed Chair in Computer Engineering, effective August 16, 1991.

Washington State University seeks an individual who has made significant contributions to the field of Computer Engineering to fill the chair endowed by the Boeing Company. The preferred candidate would be one whose primary interests are in computer systems design (e.g. networking, neural networks, high integrity systems, real time systems, computer architecture, CAD). Nominations of candidates from both industry and academia are welcome. A significant record of research funding and publications is expected.

The Department actively participates in a number of research projects that are funded by the Washington Technology Center (WTC). The WTC was established by the State of Washington in 1983 to help promote research programs in the State's research universities that will be of benefit to the industrial sector in the State of Washington. The biennial budget for the Center currently exceeds $20 million. The individual chosen, will be expected to play a leading role in the WTC Computer Systems and Software Technology program which currently has a budget of over $2.7 million for the biennium.

Recently, the Department has established an NSF Industry/University Cooperative Research Center for Design of Analog-Digital Integrated Circuits. This Center is currently sponsored by the NSF, WTC and 14 industrial companies. Depending on the area of specialty it is hoped that the Boeing Chair in Computer Engineering will actively interface with the Center personnel and its research programs.

Candidates for this position must be capable of providing leadership for the continued expansion of the Computer Engineering program within the Department. As a tenured and distinguished Professor, the individual filling the position will be expected to teach, conduct funded research, and direct Ph.D. and M.S. student research programs. An earned doctorate in engineering or a related discipline is required.

The Department of Electrical and Computer Engineering has full breadth undergraduate and graduate programs, offering the B.S., M.S. and Ph.D. degrees. Major research and academic programs exist in computer engineering, computer science, systems and controls, electronics and solid state, power, and electromagnetics. The Department recently moved into a new building with expanded research and academic facilities. There are 310 certified (junior and senior) undergraduate majors and 130 graduate (M.S. and Ph.D.) students in residence on the Pullman campus. Additionally, the department has resident undergraduate and graduate programs with 110 students in Richland, Washington. Collaborative research exists between the computer engineering faculty and mechanical engineering faculty in the area of manufacturing and robotics. Research and collaborative ties exist also with faculty at the University of Washington, the University of Idaho, and Oregon State University. Close working relationships exist between the university and the electronics industry of the Northwest.

Letter of application, vita, and the names and addresses of at least three references should be addressed to:

Dr. Yacov Shamash, Chairman
Electrical and Computer
Engineering Department
Washington State University
Pullman, WA 99164-2752

Tel: (509) 335-8148 Fax: (509) 335-3818

Washington State University is an Equal Opportunity/Affirmative Action educator and employer. Protected group members are encouraged to apply.

UNIVERSITY OF VICTORIA
Department of Computer Science

Applications are invited for at least one regular full-time faculty position to commence July 1, 1991. Applicants should have a Ph.D. in Computer Science or equivalent qualifications. The first priority is to appoint an individual with research interests in software systems, programming languages and semantics, or a closely related field.

The Department of Computer Science is in the Faculty of Engineering and will be moving to the new Engineering Office Building in December 1990. The Department offers graduate and undergraduate degrees in Computer Science and maintains active research programs in programming languages, compilers, software engineering, software development environments, distributed computing, combinatorial algorithms, theory of computation, functional and logic programming, VLSI design and test, and numerical methods. There are presently 18 faculty members and 60 graduate students, 20 in the Ph.D. program.

The Department operates a highly successful undergraduate cooperative education program and is now developing a graduate co-op option. The Department is an active participant in the Advanced Systems Institute of British Columbia with a number of ASI Fellows and ASI postgraduate student scholarship recipients.

The computing facilities available for instructional and research support currently include seven Sun 3 and Sun SPARC servers, over sixty workstations, laboratories with a variety of smaller systems, as well as the University IBM 3090/VPF system. The computing facilities across campus are fully networked while BC-Net connects UVic to the other universities and research institutions in the Province. Internet access is also available.

The city of Victoria, situated on the southern tip of Vancouver Island, has a population of over 250,000. It enjoys one of the most delightful environments in North America. The climate is temperate. Outdoor activities, from marine to mountain, are popular and can be pursued year round. Victoria is exceptionally well-endowed with cultural activities too, including theatre, opera and a symphony.

Applicants should send a curriculum vitae and the names of at least three referees to:

Dr. D.M. Miller, Chair
Department of Computer Science
University of Victoria
P.O. Box 3055
Victoria, B.C., Canada
V8W 3P6

Applicatons will be accepted until December 31, 1990. In accordance with Canadian Immigration Regulations, priority will be given to Canadian citizens and permanent residents. The University of Victoria is committed to an employment equity program. Women are strongly encouraged to apply.

Further information is available by contacting the Chair of the Department:

INTERNET: dmill@csr.uvic.ca
BITNET: dmill@uvunix.bitnet
Telephone: (604) 721-7220
Fax: (604) 721-7292

GRAND VALLEY STATE UNIVERSITY
Allendale, Michigan

Applicants invited for tenure track positions beginning Fall, 1991. Candidates must have Ph.D. in Information Systems or in Computer Science (prefer Software Engineering), with an interest in teaching at the graduate and undergraduate levels.

The department offers undergraduate majors in Computer Science and Information Systems, with masters degrees in Computer Information Systems and Software Engineering.

GVSU is located just west of Grand Rapids, the second largest metropolitan area in Michigan, offering numerous cultural and recreational opportunities. Cost of living is moderate and quality of life is high.

Send resume and 3 letters of references to: CS Search Committee, Math & CS Dept., Grand Valley State University, Allendale, MI 49401.

Consideration of applications will continue until position is filled. AA/EEO

OREGON GRADUATE INSTITUTE OF SCIENCE AND TECHNOLOGY

Would you like to work in an academic environment with an active graduate education program, but with no undergraduate teaching responsibilities? A place that encourages serious research by providing strong administrative support and excellent facilities? If so, consider joining the growing faculty of the Oregon Graduate Institute's Department of Computer Science and Engineering.

We seek both senior and junior faculty colleagues with experience in graduate education and ambitious research goals. Technical areas of particular interest include: scientific and engineering databases, distributed and concurrent computing systems, software specification and derivation, artificial neural networks and speech recognition.

OGI is located in Portland, one of the most affordable of the West Coast's beautiful cities. Portland's relaxed life style offers a setting in which both your family and your research can thrive.

For more information about OGI, please address inquiries to: Professor Richard B. Kieburtz, Chairman, Department of Computer Science and Engineering, Oregon Graduate Institute, 19600 NW von Neumann Drive, Beaverton, OR 97006, (503) 690-1150, csedept@cse.ogi.edu.

OGI is an Equal Opportunity Employer.

OHIO UNIVERSITY
Computer Science Department Faculty Position

Term position (which may possibly roll over to a tenure track position) as Assistant/Associate Professor to conduct research and to teach Computer Science graduate and undergraduate courses. Candidates must have a Ph.D. in Computer Science or equivalent (at time of appointment) with promise of excellence in research and teaching. Applicants in the area of concurrent processing and parallel programming will be given first consideration.

Departmental computing is supported by a VAX 780, a VAX 785, several AT&T 3B2's, and two microcomputer labs. Academic computing is supported by an IBM 4381, a VAX 6440, an HP3000, and several microcomputer labs.

Salary highly competitive (minima $42,000/$47,000). Application deadline: February 28, 1991. Send letter of application, curriculum vitae to and have three letters of reference sent to: Klaus E. Eldridge, Chairman, Computer Science Department, Morton Hall 423, Athens, OH 45701. Ohio University is an Equal Opportunity/Affirmative Action Employer. Women and Minorities are encouraged to apply.

UNIVERSITY OF NEVADA, LAS VEGAS
Full-Time Research Positions
Optical Character Recognition / Information Retrieval

The Information Science Research Institute at the University of Nevada, Las Vegas, invites applications for two full time positions in either OCR or Text Retrieval research. These positions are soft/non tenure-track positions at the Research Assistant or Research Associate level. Qualifications include a doctorate in Computer Science or Electrical Engineering or related area. Candidates should have demonstrated research expertise in either OCR or Text Retrieval.

ISRI has acquired and is developing a large "Ground-Truth" database for research in OCR. The database contains both the bit-mapped images and the corresponding ASCII output for 90,000 pages of English text. Adequate funds for equipment, technician support, and graduate student support are available. Several new Graduate Research Assistantships are available. Funding is available for visiting faculty positions. Interested researchers on sabbatical leave are encouraged to apply.

The College of Engineering at UNLV currently operates about 40 Sun SPARCstation1 workstations, 16 NeXT workstations, 10 color DEC VAXstation3100's, 2 Silicon Graphics IRIS workstations, a Symbolics LISP workstation, an Intel Hypercube, and several supporting file servers for faculty and student use. All equipment runs TCP/IP UNIX and is connected to the INTERNET through the campus network. Numerous PC compatible equipment is also available.

The College also houses the National Supercomputer Center for Energy and the Environment which operates a CRAY Y/MP 2:216 supercomputer. The Y/MP has 5 gigabytes of high speed disk and a 32 megaword SSD. The system is supported by a Sun SPARC 4/490 file server with 4 gigabytes of disk and a Storage Technology cartridge tape system. It is accessed locally in the Center via 10 color Sun SPARCstation1's or a Silicon Graphics IRIS workstation. The Y/MP is also on the INTERNET.

Review of applications will begin 11/1/90. Positions will remain open until filled. Salary is commensurate with qualifications and experience. To apply, send curriculum vitae, reprints of published papers, and names of at least three references to:

Dr. T.A. Nartker, Director
Information Science Research Institute
Howard R. Hughes
College of Engineering
University of Nevada, Las Vegas
Las Vegas, NV 89154

Additional information may be obtained by writing to the above address or calling (702) 739-3338 or Email (tom@cs.unlv.edu). The University of Nevada, Las Vegas, is an equal opportunity/affirmative action employer.

DARPA SPONSORED GRADUATE RESEARCH ASSISTANTSHIPS IN PARALLEL PROCESSING

The University of Maryland's Institute for Advanced Computer Studies and Center for Automation Research are soliciting applications from qualified graduate students for research assistantships in parallel processing. These assistantships are made available by funds provided by the Defense Advanced Research Projects Agency's (DARPA) Information Science Technology Office through a contract from NASA Ames Research Center. Proposals can address either fundamental issues in parallel processing (e.g. architectures, algorithms, programming languages, performance evaluation, etc.) or significant applications of parallel processing. Research proposals that have an experimental component will be given highest consideration for funding.

Awards will be made for one year of support, with possible extension to a second year based on progress made during the first year. Funds will be provided as part of the award to the student's university for salary, tuition, and benefits. The University of Maryland will provide direct travel support to an annual workshop held in conjunction with the program at which students will deliver papers describing their research and interact with leading scientists and engineers in parallel processing. If requested, students will be provided with free access to the University of Maryland's Connection Machine II.

Applications are solicited for June 1991 funding. Only one student per department will be funded.

Who Can Apply

Applicants must be:

- U.S. citizens and
- doctoral students who have completed all course work and preliminary exam requirements.

How To Apply

Applications (3 copies) should be submitted to Prof. Larry Davis, Director, UMIACS, University of Maryland, College Park, MD 20742 by January 25, 1991. an application consists of:

- a short research proposal (at most 3 single spaced typewritten pages)
- a budget endorsed by an appropriate official at the student's university (The budget should consist only of salary, tuition, benefits and overhead, if charged.)
- a letter of support from the student's faculty advisor who will serve as principal investigator
- curricula vitae of the student and advisor

For Additional Information, Contact:
Ms. Johanna Weinstein,UMIACS,University of Maryland, College Park, MD 20742-3251
301/405-6722, johanna@umiacs.umd.edu

COLGATE UNIVERSITY
Hamilton, New York 13346
Department of Computer Science
(315) 824-1000, ext. 719

We invite applications for two anticipated positions, pending administrative approval: one a tenure track position at the rank of assistant professor; the other a one-year leave-replacement position. Candidates should have or be near completion of a Ph.D. in computer science. Strong candidates in any subfield of the discipline will be considered.

Colgate is a quality liberal arts college with a first-rate computer science program. The department has five faculty with the following research interests: theory of computation and programming language semantics, computational complexity and algorithms, temporal reasoning in natural language processing, graphics and chaos, and discrete event simulation on parallel computers. The Computer Science Department has an introductory lab equipped with sixteen PCs, and an upper-level/research lab with the following equipment: a network of seven NeXT workstations, a VAX 750 running BSD 4.3 Unix, four 17-node transputer-based parallel computers, and PC/AT or NeXT workstations in every faculty office. The faculty offices and laboratory machines are connected on an ethernet. We are members of both CSNet and BitNet.

Applicants should send a resume and the names of three references to: Chris Nevison, Chairman, Department of Computer Science, Colgate University, Hamilton, NY 13346. We will consider all applications received by January 1, 1991, and applications received thereafter until the positions are filled. EO/AAE.

INDIANA UNIVERSITY-PURDUE UNIVERSITY AT INDIANAPOLIS (IUPUI)

The Department of Computer and Information Science invites applications for a tenure track position at the level of assistant or associate professor in scientific computing/software engineering. The successful candidate is expected to be a member of a new interdisciplinary group in computational science, to be developed jointly by the Department of Computer and Information Science and the Department of Mathematical Sciences.

Applicants must have an earned doctorate and strong teaching credentials A demonstrated record of research is expected at the associate professor level. Assistant professor candidates must possess exceptional research potential. Special consideration will be given to applicants with a strong background in numerical computation with expertise and interest in the development of "intelligent" software for large-scale scientific computing problems. Salary and rank will be determined by experience and qualification.

IUPUI is a rapidly growing comprehensive urban university with over 26,000 students. The university offers competitive salaries and provides excellent fringe benefits. Indianapolis is a large, active metropolis and has available a dynamic industrial base conducive to joint research and consulting with our faculty. The six-hospital Medical Center at IUPUI is yet another source of interdisciplinary research for the Department.

Applications and inquiries should be addressed to Prof. Raymond C. Y. Chin, Chair, Department of Computer and Information Science, IUPUI, 1201 East 38th Street, Indianapolis, IN 46205-2868. Deadline for applications: December 1, 1990. Late applications will be considered until position is filled. IUPUI is an Equal Opportunity/Affirmation Action Employer. Women and minority candidates are encouraged to apply.

WESTERN MICHIGAN UNIVERSITY
Computer Engineering and Design

The Department of Electrical Engineering invites applications for an anticipated tenure-track faculty position at the Assistant/Associate/Full Professor level. Preferred applicants must have an earned Ph.D. in Computer Engineering or the equivalent and specialize in computer engineering and design. Responsibilities include graduate and undergraduate teaching, curricular leadership, and research. Western Michigan University is located in Kalamazoo, Michigan and has approximately 26,000 students; it is one of the state's five graduate intensive universities and is designated a Carnegie Doctoral I university. The Department offers two EAC/ABET accredited undergraduate degrees (Computer Systems Engineering and Electrical Engineering) and a growing graduate program. We are looking for an experienced computer engineering faculty member who enjoys teaching and would like to participate in building a design oriented graduate program in computer engineering. Please send a detailed resume, including a statement of citizenship, and names of three references to: Dr. Thomas F. Piatkowski, Chair, Department of Electrical Engineering, Western Michigan University, Kalamazoo MI 49008-5066 or piatkowski@gw.wmich.edu. Applications will be accepted until the position is filled. Western Michigan University is an affirmative action employer/equal opportunity institution.

SENIOR SOFTWARE ENGINEER

Senior Software Engineer needed to design and develop voice digital processing systems to be used in voice response telephone automation and call handling utilizing IBM/PC and PS/2 microcomputers, "C," Assembler and SmallTalk languages. Diagnose, define and resolve software problems and provide hardware performance tuning and maintenance service. Requires a Bachelor's degree in Electronics Engineering/Computer Science/Computer Engineering or its equivalent and two years experience in job offered or two years directly related software engineering experience. Requires at least one year experience working with Assembler and C languages. 40 hour work week. $40,000.00 per year. Apply at the Texas Employment Commission, Dallas, Texas, or send resumes to the Texas Employment Commission, TEC Building, Austin, Texas 78778. Job Order #5515164. Ad paid by an Equal Employment Opportunity Employer.

UNIVERSITY OF MINNESOTA
Supervising Systems
Software Programmer

BA/BS in business administration, industrial engineering or related field with five or more years experience in system analysis and computer programming of which at least one year must be at a responsible supervisory level. Three years of experience in Hewlett Packard environment dealing with research, clinic operations, and office management. Varied tasks of both managerial and technical nature. Strong people skills required. Responsibilities: planning, contingency recovery, development, know Cobol, micro computers and HP preferably in a healthcare environment. Send resume and three references by November 10, 1990 to:

Attention: Joan Felty
Department of Family Practice & Community Health
University of Minnesota
Box 381 UMHC PWB 6-240
516 Delaware Street S.E.
Minneapolis, Minnesota 55455

The University of Minnesota is an equal opportunity educator and employer and specifically invites and encourages applications from women and minorities.

COMPUTER SCIENCE RESEARCHERS

The Centre de recherche informatique de Montreal (CRIM) is a non-profit corporation focusing on excellence in research and development and on the transfer of information technologies and their applications.

CRIM is seeking Computer Science Researchers to join its research groups.

RESPONSIBILITIES

Participation in joint university-industry research projects. Domain: software engineering. Specialization: automatic support and assistance for software design and reuse; reverse engineering. Knowledge of Artificial Intelligence is an asset.

QUALIFICATIONS

- The applicant must hold a Ph.D. in computer science with sufficient publication. Industrial experience is sought.
- Priority will be given to Canadian citizens and landed immigrants.
- Knowledge of French is an asset.

Interested candidates may apply by sending a resume before January 15, 1991, to:

Dr. Jacqueline Bourdeau
Centre de recherche informatique de Montreal
3744, Jean-Brillant St., Suite 500
Montreal (Quebec) H3T 1P1

TEACHER
Mathematics and Computer Mathematics, Secondary School

Minimum two years experience and BA Degree. Will teach algebra, concepts of geometry, trigonometry, pre calculus and computer mathematics including teaching, PASCAL programming algorithm development and advanced programming techniques. Salary $39,200.00 per year. Job site and interviews: Los Angeles, California. Send this ad and your resume to 5405. Commonwealth Ave., Los Angeles, CA 90020.

UNIVERSITY OF NOTRE DAME
Chairperson
Department of Computer Science and Engineering

The University of Notre Dame invites applications for the Chair of the recently established Department of Computer Science and Engineering in the College of Engineering. This position is a unique opportunity to provide significant input in establishing direction for the initiation and growth of programs in this department. Persons with established records in education and scholarship who would welcome the challenge of developing a new department and its programs are encouraged to apply. The faculty of the department is expected to determine the curricula leading to graduate and undergraduate degrees in computer science and in computer engineering.

The Department of Computer Science and Engineering has twelve tenure-track faculty positions. The Department of Electrical Engineering and its present graduate and undergraduate options of emphasis in computer engineering are expected to provide the foundation for the programs in the new department. It is anticipated that the Department of Computer Science and Engineering will have close cooperative ties with the Department of Electrical Engineering. Department of Mathematics, and other academic units.

Qualified applicants are encouraged to submit a resume and letter of interest to:

Dr. Panos J. Antsaklis
Department of Electrical Engineering
University of Notre Dame
Notre Dame, IN 46556

For additional information, please contact the Dean's office at (219) 239-5534.

The University of Notre Dame invites applications from all qualified persons without regard to sex, ethnic origin, religious preference or physical impairment.

PENN STATE
Director of Computer Engineering

Applications/nominations are invited for the position Director of Computer Engineering in the Department of Electrical and Computer Engineering at The Pennsylvania State University available Spring or Fall Semester, 1991. Candidates should have a Ph.D. in Electrical/Computer Engineering or a related discipline, an established reputation in Computer Engineering, strong leadership qualities, and ability to attract research funding. It is anticipated that the position will be at the rank of Full Professor with tenure. The Department of Electrical and Computer Engineering at Penn State currently has over 50 faculty, 800 junior and senior level students, and 280 graduate students. The Computer Engineering Program is within the Department of Electrical and Computer Engineering. About 12 faculty members, 100 junior and senior level students, and 50 graduate students belong to this program, which awards B.S., M.S. and Ph.D. degrees in Computer Engineering. Research activities currently exist in Parallel and Distributed Processing, Interconnection Networks, Fault Tolerant Computing, Image Processing and Computer Vision Database Systems, VLSI, and Computer Communication. Applications received by December 31, 1990 will be assured of consideration. However, applications will be considered until positions are filled. Please send resume and cover letter, with names, addresses and phone numbers of at least three references to: Faculty Recruiting Committee, Department of Electrical and Computer Engineering, Box CM-D, 129 EE East, The Pennsylvania State University, University Park, PA 16802.

An Affirmative Action/Equal Opportunity Employer. Women and Minorities Encouraged to Apply.

UNIVERSITY OF ALBERTA
Department of Computing Science

Applications are invited for a tenure-track position at the Assistant Professor level in the areas of computer graphics and/or human interfaces. Responsibilities include research as well as teaching at both the graduate and undergraduate levels.

This young and energetic department has recently undergone significant expansion and consists of 35 academic and 29 support staff. Current hardware support for research includes a local area network consisting of a MIPS M/120 and Silicon Graphics AD340S computer connecting over 60 Sun Workstations and a large number of terminals. Current computer graphics facilities include two Silicon Graphics workstations, a Data Glove, a head-mounted display, a single frame video recorder and two colour Sun workstations. The department recently acquired a Myrias SPS-2 massively-parallel computer to use in collaborative research with Myrias Research of Edmonton. Well-supported laboratories also exist for AI, database, distributed systems, image processing, programming languages and robotics research. Access to a Cyber 205 is available.

The current minimum salary range is $C36,910 with the appointment level being commensurate with qualifications and experience. Send *curriculum vitae*, the names of three references and up to three reprints or copies of important publications. New Ph.D's should also include a copy of their transcript. Apply to:

Dr. Paul G. Sorenson
Department of Computing Science
University of Alberta
Edmonton, Alberta, T6G 2H1
Canada

In accordance with Canadian Immigration requirements, priority will be given to Canadian Citizens and Permanent Residents.

The University of Alberta is committed to the principle of equity employment.

DREXEL UNIVERSITY
Department of Mathematics and Computer Science

The Department of Mathematics and Computer Science seeks computer scientists for anticipated visiting and tenure-track positions including assistant professor through professor. Research interests should include computer algebra, parallel and distributed computing, numerical analysis, computer graphics, software engineering, or mathematics and computer science education (role of computers in undergraduate curricula; design of software, systems and instructional methods).

The department currently has 30 tenure-track faculty in mathematics, statistics and computer science offering B.S., M.S. and Ph.D. programs. Research activities also include discrete mathematics, functional analysis, applied analysis, computational statistics and artificial intelligence. A departmental network connects a Sequent Symmetry S27, Sun workstations and servers, and Apple Macintoshes and provides access to NSFNET and Drexel's IBM 3090-VF.

Drexel University, founded in 1891, consists of six colleges with about 12,000 students. Undergraduate education is distinguished by two features: a comprehensive cooperative education program and the Drexel Microcomputer Program—every undergraduate student must have personal access to an Apple Macintosh.

Send resume and names of three references to: James C. T. Pool, Mathematics and Computer Science, 206 Korman Center, Drexel University, Philadelphia, PA 19104. Application deadline is November 30, 1990 or until approved positions are filled.

Drexel University is an equal opportunity/affirmative action employer and encourages applications from women and minority candidates.

UNIVERSITY OF NEBRASKA-LINCOLN
Computer Science & Engineering
Faculty Positions

Applications are invited for one or more tenure-track positions in CSE or for joint appointments with Electrical Engineering or UNL Research Centers starting August, 1991. Rank and salary will be commensurate with experience with hiring preference at the Assistant Professor level. Women and minorities encouraged to apply.

Areas targeted for expansion are: (1) Database, (2) Software Engineering, (3) Operating Systems, (4) Networks and Communications, (5) Compilers and Programming Languages, (6) Artificial Intelligence and Neural Networks, (7) VLSI, and (8) Computer Architecture and Parallel Processing. Outstanding applicants in other areas will also be considered.

The department has 18 full-time faculty active in research and offers B.S., M.S., and Ph.D. programs. It has 60 graduate and 460 undergraduate students.

UNL has many computing resources linked by a comprehensive campus-wide network. It is the lead institution in the NSF-funded regional network, MIDnet and a node on the NSFnet backbone, and has state supported centers including the Center for Communication and Information Sciences and the Center for Technology Development and Decision Sciences.

Send resume, names of three references and letter describing current teaching and research interests by January 6 (or until suitable candidates apply thereafter to:

Alvin J. Surkan, Search Chair
Department of Computer Science & Engineering
UNIVERSITY OF NEBRASKA-LINCOLN
Lincoln, Nebraska 68588-0115

Affirmative Action/Equal Opportunity Employer

1951-1991
40 YEARS OF SERVICE
IEEE COMPUTER SOCIETY

IEEE COMPUTER SOCIETY
Membership / Subscription Application

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

BENEFITS

Computer
You automatically receive *Computer* with membership. Written, reviewed, and refereed by experts, it features survey and tutorial articles covering the entire computer field, and departments such as new products, product reviews, standards, and a reader forum called "The Open Channel." (monthly).

Technical Committees
Participate in one or more of our 33 technical committees — networks of professionals with common interests in specialty areas within computer hardware, software, and applications.

Standards Working Groups
Participate in the development of the more than 100 standards projects currently sponsored by the society in such diverse areas as software engineering, local area networks, microprocessor buses, design automation, programming languages, and standards definitions.

Computer Society Press Books and Videos
Receive discounts of up to 50% on over 700 titles covering a broad spectrum of computer science topics such as networking, communications, advanced systems, image processing, security, artificial intelligence, and visualization. Over 120 new products are published annually.

Conferences and Tutorials
Choose from more than 100 conferences annually, ranging from large industry-oriented conferences replete with exhibits to small, highly interactive workshops. Members receive special low rates.

Schedule of Fees

To join: see item 1, 2, or 3.
To subscribe: see item 4.

Membership dues and periodical subscriptions are annualized to, and expire on, December 31. Pay full- or half-year rate depending on date of receipt by the Computer Society as indicated below.

	Half Year Mar 1-Aug 31	Full Year Sept 1-Feb 28
1 I don't belong to the IEEE and I want† to join just the Computer Society	☐ $27.00	☐ $ 54.00
2 I don't belong to the IEEE and I want to join both the Computer Society and the IEEE*		
I reside in Region 1-6 (United States)	☐ $52.50	☐ $105.00
I reside in Region 7 (Canada)	☐ $48.50	☐ $ 97.00
I reside in Region 8 (Europe, Africa, or the Middle East)	☐ $48.00	☐ $ 96.00
I reside in Region 9 (Latin America)	☐ $44.50	☐ $ 89.00
I reside in Region 10 (Asia and Pacific)	☐ $43.50	☐ $ 87.00

*ACM members who join both IEEE and the Computer Society may deduct $5 off the full-year rate; $2.50 off the half-year rate

3 I already belong to the IEEE and I want to join the Computer Society	☐ $11.00	☐ $ 22.00

IEEE Member Number _____

4 OPTIONAL PERIODICALS for new or current members

	issues per year		
IEEE Computer Graphics and Applications	6	☐ $12.00	☐ $ 24.00
IEEE Design and Test	4	☐ $11.00	☐ $ 22.00
IEEE Expert	6	☐ $10.00	☐ $ 20.00
IEEE Micro	6	☐ $10.50	☐ $ 21.00
IEEE Software	6	☐ $12.50	☐ $ 25.00
Transactions on:			
Computers	12	☐ $12.00	☐ $ 24.00
Knowledge and Data Engineering	4	☐ $ 7.50	☐ $ 15.00
Parallel and Distributed Systems	4	☐ $ 7.00	☐ $ 14.00
Pattern Analysis and Machine Intelligence	12	☐ $12.00	☐ $ 24.00
Software Engineering	12	☐ $11.00	☐ $ 22.00

Total amount remitted with this application $ _____

DC residents add sales tax to optional periodicals.

☐ Checks accepted in Belgian, British, German, Swiss, Japanese, or US currencies.
Checks must be drawn on a bank in the country of origin of the currency

PRICES EXPIRE 12/31/91

☐ Visa ☐ Master Card ☐ American Express

Charge Card Number (Minimum Charge $10.)

Mo. / Yr. — Exp. Date

I hereby make application for Computer Society membership and, if elected, will be governed by IEEE's and the society's constitutions, bylaws, and statements of policies and procedures.

MAILING ADDRESS

Full signature — Date

☐ Male ☐ Female

First name — Middle initial(s) — Last name — Date of birth

Street address — City — State/Country — Zip

EDUCATION (highest level completed)

Course — Degrees received — Date

Name of educational institution

ENDORSER (an **IEEE** member, Senior Member, or Fellow who knows you professionally)

OCCUPATION

Title or Position — Endorser's signature

Firm name — Name (print in full) — IEEE Member No.

Firm address — Street address

City — State/Country — Zip — City — State/Country — Zip

Return to: IEEE Computer Society, 10662 Los Vaqueros Circle, PO Box 3014, Los Alamitos, CA 90720–1264 USA.
Residents of Europe mail to: IEEE Computer Society, 13, Avenue de l'Aquilon, B-1200, Brussels, BELGIUM.
Asian/Pacific residents mail to: IEEE Computer Society, Ooshima Building, 2-19-1 Minami-Aoyama, Minato-ku, Tokyo 107 JAPAN.

PC1190

BOOK REVIEWS

Editor: Guy Johnson, Department of Information Technology, Rochester Institute of Technology, 1 Lomb Memorial Drive, Rochester, NY 14623.

Coordinating User Interfaces for Consistency

Jakob Nielsen, ed. (Academic Press, San Diego, Calif., 1989, 142 pp., $24.95)

Anyone working toward consistency in user interfaces will find this book interesting, including human factors professionals and software engineers working on user interface requirements, design, or implementation. While most of the collected papers discuss experiences and insights from the makers of commercial software, implementors of user interfaces for military systems will also benefit from the contents.

Readers with a user interface background could use this book for self-instruction. It is not applicable as a course text, but it could serve to illustrate user interface production in a commercial software environment.

The book is divided into five parts, with a total of eight papers. Earlier versions of two papers have appeared elsewhere. The first part is an executive summary by the editor. The second part contains three papers on the dimensions of consistency, its economic implications, and the need to coordinate the various aspects of the user interface. The four papers in the third part are case studies from major companies that have instituted programs to ensure the consistency of their own products and of third-party products. The fourth part has only one paper, which describes a technological solution to the consistency problem in the form of tools to generate user interfaces. A bibliography comprises the fifth part.

One of the papers in the second part is worth the price of the book. "A Cost Benefit Analysis for Corporate User Interface Standards: What Price to Pay for a Consistent Look and Feel?" is clearly organized, well written, and includes excellent graphics. Author Daniel Rosenberg identifies companies that have announced plans to standardize user interfaces within and across product lines, and he describes the design elements of a user interface and the scope of consistency for software requirements and design. Rosenberg identifies the potential benefits of consistency, where money can be saved, and how to calculate the break-even point for implementing a user interface that is consistent across products. He discusses marketplace variables, technical factors, and legal issues, and he concludes that the development of corporate user interface standards is justified technologically and financially, and that no significant financial, technical, legal, or market factors limit the concept's usefulness.

The paper titled "Developing the XUI Style" describes Digital Equipment Corporation's design methods and underlying philosophy in the DECwindows program. The methods involve technical and organizational aspects of developing a consistent user interface style, and the techniques include widespread use of electronic mail and conferencing and sharing of user interface prototypes.

A novel aspect of XUI's development was the acceptance of graphical specifications for the user interface. This is an excellent way to specify an interface. Unfortunately, many other organizations have resisted changing the software development life cycle and the associated specification medium. For example, the development of military systems is limited by the need to express specifications in written form. Furthermore, the resulting requirements can be interpreted in varied ways.

For example, TRW recently spent two years developing and refining a prototype user interface. Several scenarios of user interactive activity were simulated using a typical hardware configuration (color display, keyboard, trackball) and a software tool that provided the ability to simulate real-time events, prepare event-driven display panels, and process user dialogue sequences. The user interface developed from this effort was captured in written specifications, along with a number of representative display panels. However, the display panels were eliminated from the specifications, since such displays are considered too design oriented and are not accepted as a means to specify testable, traceable requirements. Consequently, the final user interface will be based only on the written requirements and could differ considerably from the one seen and refined by the system engineers and users. Acceptance of graphical specifications for the user interface would minimize such problems.

The book has several recurrent themes as well as themes that the reader can infer and develop. One recurring theme is an approach called consistency by explanation and example. In this approach, a user interface style guide is prepared, along with an extendable toolkit implementing the style. A key application is then built according to the style guide and using the toolkit. This approach was successful for Apple and Digital and is much discussed in two papers by IBM researchers.

Another theme developed in two of the papers is that there are two dimensions of consistency: conceptual consistency and consistency due to a common look and feel. Conceptual consistency uses a consistent navigation model to define action relationships and a consistent data model to define how various types of information are displayed. For consistency based on a common look and feel, standard appearance elements provide the look, and consistent interaction rules provide the feel.

The book contains a few errors, but corrections were either evident or could be inferred from correct portions of the text. I never needed another reference to resolve an error.

Not all the papers are as interesting as the ones I mentioned above, but most researchers and practitioners involved with user interface consistency should find most of the the material very useful.

Andrew J. Paroczai
TRW Space & Defense Sector

Measuring Software Design Quality

David N. Card and Robert L. Glass (Prentice Hall, Englewood Cliffs, N.J., 1990, 130 pp., $30)

Measurement now plays only a small role in software development, confined mostly to cost estimation. This book describes other roles for software measurement in design engineering, quality control, and technology evaluation. Most measurement books, such as *Software Engineering Metrics and Models* by S.D. Conte et al. (Benjamin/Cummings, 1986), take an encyclopedic approach, measuring everything that can be measured. *Measuring Software Design Quality*, however, proposes a limited set of measures focusing on design quality.

The authors' approach holds that software design is more critical than software implementation, and that design quality (or lack thereof) has consequences throughout the software life cycle. This perspective on software development and design leads to a better understanding of what comprises good design and challenges some widely accepted design practices.

This book does not describe a new design methodology. Instead, it describes how measurement works given a specific set of methods to ensure a good design. (I agree with the authors' statement, "Without measures, methods are not engineering.") You might be surprised that this book covers two delicate subjects like measurement and design in only 130 pages, but the book's size relates to both its good and its bad aspects.

The book is organized in four parts. Part I provides background material, emphasizing software complexity, modularity, and producibility as key factors in a good software design. Part II then describes the evolution of the authors' design complexity model. This section discusses milestones in the development of software complexity measures and examines modularity as a measurable factor in design quality. This part also contains comprehensive software complexity models derived from the authors' experience.

Part III identifies other principal uses of measurement, such as cost estimation, quality control, and process improvement. Finally, Part IV provides additional details and supporting materials for topics introduced elsewhere in the book.

The book is targeted at practitioners, software engineers, and software managers, who generally do not like to read long, verbose books. This audience could be attracted by the book's size as well as by the fact that it is well written and organized. However, the book requires that readers have a very strong background in software engineering metrics and models, limiting its use for self-instruction. Also, the book's size limits detailed discussions of important topics, such as system complexity models and unit complexity models (although the book is not superficial). Lastly, the book relies too heavily on references for a book that is not intended for an academic audience.

Readers who are familiar with the theory of software measurement can expect to come away from this book with the ability to operate a successful measurement program by selecting measures that respond to the organization's needs. The book could also supplement a graduate course text on software engineering measurement techniques. This is not an introductory book, but it could be used to show a pragmatic approach to this discipline.

Considering that, in very few pages, readers are prompted to think about ways to improve their measurement and management of software development, this book is well worth reading.

Paolo Spiller
ARTIS, Italy

Building a Secure Computer System

Morrie Gasser (Van Nostrand Reinhold, New York, 1988, 299 pp., $39.95)

Given the development of computers and open networks, the security and control of computer and information systems is becoming more vital than ever. Gasser has been involved in computer security since 1971 and is one of the best-qualified people to offer their knowledge to the computer security community. His goal in this practical and accurate book is to make the purpose of security understandable.

Gasser asks such questions as: What is computer security? What are the evaluation criteria? Why aren't systems secure? and Why is computer security difficult? He takes into account the evolution of computer security — which used to refer only to the physical protection of information and hardware protection against failure — and emphasizes the problem of guarding against unauthorized access, and securing privacy, secrecy, and integrity. For given problems, he addresses the technical requirements in terms of security policy, assurance, accountability, and documentation. Gasser also acknowledges that security is difficult and sometimes considered a nuisance associated with reduced performance and increased costs.

Specific topics include design techniques, access controls, multilevel security, covert channels, Trojan horses and viruses, and networks. Gasser cleverly avoids going into detail on applications that are already covered in other reference books, concentrating instead on subjects that are poorly documented elsewhere. His explanations are explicit, and he gives fundamental references for readers who want to know more. Each chapter contains references, and there is a final general literature guide.

I read the book with real pleasure because of its clarity, its homogeneity, and its appropriate and fundamental bibliography. Gasser makes the reader feel comfortable with the subject, and he pays considerable attention to choosing appropriate technical solutions to practical problems, making the book useful for anyone who uses or designs a computer system. For all these reasons, I recommend the book to the computer security community, to computer professionals and students with no particular background in the area, and to instructors for courses or seminars on secure computer systems or computer security.

Daniel Guinier
CNRS and IREPA, France

Digital Image Processing and Computer Vision

Robert J. Schalkoff (John Wiley & Sons, New York, 1989, 489 pp., $65.95)

Schalkoff has been quite busy. In addition to this book, his *Artificial Intelligence: An Engineering Approach* was just published by McGraw-Hill. (However, *Digital Image Processing and Computer Vision* is a much handsomer book; the design of the other is quite distracting, and I'm glad not to review it.)

In the preface, Schalkoff reveals an ambitious agenda for the book. He states that the book's model-based approach distinguishes it from "others that tend to be illustrated encyclopedic summaries of the plethora of ad hoc image processing and computer vision approaches." He also states that "the most difficult aspect of writing [a] text is avoiding both superficiality and an *n*-volume text." (I thoroughly agree.)

He points out "...the underlying complexity of many image processing and computer vision tasks... often considered by newcomers (and some managers) as trivial. Thus I suspect that practicing engineers and scientists will find the presentation somewhat more practical and perhaps a bit less esoteric than might be found in other texts . . ."

So, how well does Schalkoff fulfill this ambitious agenda? Like all reviewers, I first examined the blurbs on the dust jacket, where I found the statement: "Rank order filtering, including median filtering, is new and does not appear anywhere else in book form." I decided to look into that one. Following the index listing to page 137, I found the heading "Rank (and Median) Filters" and a fairly useful, detailed, and interesting review of median filters. Alas, there was nothing more; I expected at least a discussion of extremal filters, if not the pseudomedian, to justify the jacket blurb. However, Schalkoff does manage in just a few pages to present a precise, accurate, and rigorous definition of the median filter, with one graphic illustration to show where it would be useful (although he does not offer any examples of edge distortion to illustrate possible pitfalls for the unwary).

I next looked at the bibliography at the end of the chapter. There is no guide to that excellent and voluminous list except for the references in the text. Several significant items are not even mentioned in the text. To compound matters, there is no index of authors or citations, only the general index of topics. (Incidentally, I couldn't figure out the ordering rules for end-of-chapter references. In some chapters they are alphabetized, in others they are in seemingly random order.)

Finally, I examined the problems at the end of the chapter. The two problems on median filters were well chosen to reinforce understanding of the chapter. Overall, a rather mixed first impression.

The author has divided the book into three topical areas: early vision (preprocessing) concerns and algorithms, high-level visual processing and understanding, and image processing and computer vision hardware.

The first area is covered in three chapters: "A Geometrical Model for Imaging and Some Applications," and "Image Grey-Level Modeling and Early Processing Fundamentals" Parts I and II, which cover transforms and sampling, and enhancement, restoration, and conversion, respectively.

The second area also receives three chapters: "Image Motion: Modeling, Detection, Interpretation, and Understanding — Dynamic or Time-Varying Image Analysis," and "Image Analysis" Parts I and II, these last two covering pattern recognition, segmentation, and low-level descriptions, and hierarchical image analysis, higher dimensional modeling, and image-based knowledge manipulation, respectively.

Finally, the third part has only one chapter: "Practical Image Processing Concerns."

Intended as a teaching tool, each chapter begins with a simplified model of the concepts to be introduced. The author expects readers to have backgrounds in or exposure to linear algebra, probability and random processes, some geometry, and linear systems and filtering concepts. The appendixes contain some concise but useful information on these topics.

Given these prerequisites, it is amazing how much information can be imparted, subject to severe pruning of the material, especially when supplemented by the problems at the end of each chapter. Unfortunately, the text does not adequately support the example figures, resulting in some mysterious image-transformation sequences. A typical example is a figure intended to illustrate the "trade-off between image precision and computational expense" in some nonlinear image coordinate transformation with interpolation. The text says only that the figure "compares the visual effects of interpolation on a particularly difficult image, namely one with significant and regular high frequency characteristics." The figure caption reads: "Interpolation function examples: (a) Original binary grid image; (b) Binary image with Nene interpolation; . . ." Nothing is stated about the resolution of the "binary grid image." The coordinate transformation is rather nonlinear but is left undefined. The interpolations are:

- nearest neighbor ("Nene," although "NN" is more common),
- average value of the four nearest neighbors,
- rank 2 (second highest intensity of four nearest neighbors) (Ah ha, there's the rank filter, though this mention of it is not listed in the index, and rank filters are not discussed elsewhere), and
- bilinear.

Another problem here is that the pixel sizes in the examples seem much coarser than in the original image. This type of problem occurs throughout the book.

In Chapter 3, the author addresses image transformations by introducing hierarchical categories that promise to unify the most diverse approaches:

(1) Stochastic (probabilistic) models with no geometric dependence. In other words, point transformations, such as "image intensity histograms for enhancement purposes."

(2) Deterministic models that rely on local grey-value variations (in other words, convolutions).

(3) Stochastic models that rely on local grey-value dependence, such as cooccurrence matrices.

(4) Deterministic models that rely on global image properties, both geometric and intensity based.

Aside from the quibble that histograms are global entities — even if histogram modification is indeed a point operation —this is a fascinating pedagogical device.

The author approaches transforms from the viewpoint of image basis functions, so that he introduces discrete transforms directly rather than moving from the continuous case to the discrete. (Indeed, he introduces continuous Fourier transforms later as a "digression.") He also introduces sampling theory "from the transform viewpoint," another pedagogically useful approach. FFT is introduced only as a single problem at the end of the chapter, with a single book reference. The chapter also discusses Walsh-Hadamard and slant transforms. I couldn't find the Karhunen ve (K-L) in the index.

In Chapter 4, the author emphasizes the statement: "It is difficult to underestimate the utility of reliable edge extrac-

tion." However, his actual discussion of the topic is lacking. After a brief discussion of noise filters — namely, the 3×3 averaging convolution and the median — he goes into spatial gradients and their approximations, zero crossings of second spatial derivatives (Marr-Hildreth), surface fitting (Frei-Chen), and laplacian of gaussian (LOG). The difference of gaussians (DOG) is treated in exactly one sentence (in which it is not explicitly named), while the evocative name "Mexican hat" is ascribed to the LOG operator. Although the author mentions the directional property of the zero crossings, he cites Haralick's algorithm (zero crossings of the second directional derivative) only in the end-of-chapter references. The author promises in the preface to explain "the 'why' (as opposed to the 'how')," but nowhere could I find a discussion of why someone would choose one of these edge-detection algorithms over another.

The chapter closes with a brief, unsatisfactory discussion of image compression, including thresholding, dithering, and run-length encoding. There are also a few pages on image reconstruction from back projections. The topic is illustrated by Fortran listings of a simple binary image, "reconstructed" from its horizontal and vertical projections. The other chapters are just as disjointed, although they offer solid discussions of the material.

Chapter 8 covers practical image processing concerns and starts with a discussion of imagers. A brief and inaccurate discussion of the old NTSC black-and-white standard ends with the statement: "The serial standard we have investigated is but one of a number of existing standards (such as PAL and SECAM)." That is the extent of the discussion of color standards. Also, the comparison of vidicon and CCD sensors does not mention the problem of geometric distortion in the former, and at one point the vidicon target is incorrectly called a "phosphor."

In spite of his intentions, the author has not entirely avoided superficiality. The book's organization is noteworthy, and the presentation is concise and accurate, but the individual chapters are somewhat disorganized. However, the choice of topics is excellent and the problems at the end of each chapter are well designed. Given a competent instructor to supplement (and sometimes to correct) the text, it could make a good upper undergraduate text. It could also serve as an introductory text for practicing engineers who have not encountered digital image processing before. But graduate students and knowledgeable professionals will be better served by Levine's *Vision in Man and Machine* (McGraw-Hill, 1985) or Gonzalez and Wintzes' *Digital Image Processing* (Addison-Wesley, 1987).

Morton Nadler
Virginia Polytechnic Institute and State University

The Design and Analysis of Parallel Algorithms

Selim G. Akl (Prentice Hall, Englewood Cliffs, N.J., 1989, 401 pp., $44)

This is not just another book on the flourishing subject of parallel computing. Akl is one of the few authors who can transform such hard material into beautiful and attractive science.

In surveying existing parallel algorithms, with emphasis on design methods and complexity results, Akl has undertaken a much broader subject here than in his first book, *Parallel Sorting Algorithms* (Academic Press, 1985). The computational problems he presents can be grouped into three classes:

- Class 1 (Chapters 2-6) — sorting, searching, and related problems such as selection, merging, and generating permutations;
- Class 2 (Chapters 7-9) — numerical problems such as elementary matrix operations, fast Fourier transforms, and some basic linear algebra kernels; and
- Class 3 (Chapters 10-14) — problems arising in various application areas, such as graph theory, computational geometry, tree traversals, optimization problems, and the bit-complexity of computing.

Chapter 1 is the root of the book, from which the reader can skip to any chapter. It is much more than a standard introduction, since it also describes Akl's framework for expressing and analyzing algorithms. The chapter starts by describing the various computational models used in the book: shared-memory SIMD computers, interconnection-network SIMD computers, and MIMD computers. Akl then details the criteria for evaluating algorithms: running time, number of processors, cost, and trade-offs. Finally, he presents a simple formalism for expressing parallel algorithms, consisting of a Pascal-like description extended with standard parallel constructs. Chapter 1 sets the ground rules so well that readers can then read any chapter.

For each problem under study, Akl describes an algorithm (sometimes sequential, sometimes parallel from the start), a complexity analysis for the target computer model, several algorithmic improvements to reach optimum results, and references to recent literature for further investigation.

For example, Chapter 3's discussion of merging starts with an odd-even network and moves to shared-memory models: first CREW (concurrent read, exclusive write) merging (with simultaneous reading allowed), then EREW (exclusive read, exclusive write) merging. The resulting algorithm is not optimal, so the next section refines it and designs a better one. This unified framework for expressing and evaluating algorithms, together with Akl's refined classification of the target architectures, is really the feature that makes this book a pleasure.

Akl's treatment of Class 1 problems has no counterpart in the literature and by itself makes the book worth buying. I also appreciated the material on Class 3 problems, and I particularly recommend the chapters on computational geometry and traversing combinatorial spaces.

Some of the chapters on Class 2 problems are weaker. For example, the solution of linear systems of equations calls for more than just a Gauss-Jordan algorithm for SIMD computers and a Gauss-Seidel algorithm for MIMD computers. Modi's *Parallel Algorithms and Matrix Computation* (Oxford University Press, 1988) offers a more in-depth treatment of this topic. However, I did like the clean presentation of parallel discrete and fast Fourier transforms in Chapter 9.

The book focuses on theoretical models and complexity rather than on real-life implementations. In other words, algorithms are evaluated according to their order of magnitude: $O(\log \log n)$ is better than $O(\log n)$, which is better than $O(n)$, and so on. But this is not a drawback; every discipline needs a foundation.

The book seems to be oriented toward graduate or upper undergraduate students, but experienced researchers should also find it useful for its careful analysis of classic algorithms. If you teach parallel algorithms, regardless of the orientation of your lectures, you need this book.

Yves Robert
Ecole Normale Superieure de Lyon, France

Current contents CS MAGAZINES

October 1990 *IEEE Micro*

Guest Editors' Introduction: The Maturing of DSP
Stephen A. Dyer and Richard J. Higgins

Programmable DSPs: A Brief Overview
Edward A. Lee
Now that DSPs appear in so many applications, it's time to take a look at their history and the ways they may impact the future.

Merging Data Converters and DSPs for Mixed-Signal Processors
Henry Davis, Robert Fine, and Denis Regimbal
It may not be brand new, but the use of Sigma-Delta technology to create a mixed-signal processor is hot off the press!

Gabriel: A Design Environment for DSP
J.C. Bier et al.
An experimental software system manages application development from conceptualization and experimentation to deployment in real-time hardware.

Designing a Custom DSP Circuit Using VHDL
Krishna A. Kumar and Brian Petrasko
When design size and complexity boggles your mind and the usual tools don't help, turn to flexible VHDL. You'll enjoy the control it gives you.

A TMS3020C25-Based Multirate Filter
Ralph Chassaing, Wayne A. Peterson, and Darrell W. Horning
Improved filters support DSP applications that span several frequency ranges. One time-saving project makes use of popular, second-generation equipment.

Special Feature: Parallel CRC Generation
Guido Albertengo and Riccardo Sisto
The trend toward higher and higher data rates makes a parallel circuit for CRC encoding intriguing.

October 1990 *IEEE Expert*

Setell: The Strategy Expert for Telecommunication Investments
Ahti Salo and Raimo P. Hämäläinen
Integrating economic and behavioral factors with complex engineering knowledge.

Expert Design of Local Area Networks
Stefano Ceri and Letizia Tanca
Using a rule-based expert system to configure distributed information systems.

Peirce: A Tool for Experimenting with Abduction
W.F. Punch III et al.
Building abduction systems with domain-tuned assembly strategies.

An Application of Explanation-Based Learning to Protocol Conformance Testing
C. Geldrez et al.
Automating test selection and construction as effectively as possible.

Railcar Distribution at the French Railways
Pierre Levine and Jean-Charles Pomerol
Operational research or expert system?

October 1990 *IEEE Design and Test*

Guest Editorial: High-Level Synthesis of Digital Circuits
Giovanni De Micheli
High-level synthesis systems are destined to emerge as a key technology in digital design. Their major advantage is the freedom they give to designers to explore architectural trade-offs.

From Behavior to Structure: High-Level Synthesis
R. Camposano
In this tutorial look at the technology, the author describes how synthesis bridges the gap between behavioral specifications and hardware structure by automatically generating a circuit description from a netlist.

An Optimizer for Hardware Synthesis
J. Bhasker and Huan-Chih Lee
Modern compilers offer a well-developed technology for analyzing and optimizing programs. With a few modifications, these techniques can be applied to hardware synthesis to attain maximum speed with the minimum commitment to resources.

The Olympus Synthesis System
Giovanni De Micheli et al.
Olympus, a synthesis system for digital design developed at Stanford University, is an integrated set of tools for multilevel synthesis, technology mapping, and simulation. This practical system has been tested against benchmark circuits and has been used in the design of three ASIC chips.

Neural Net and Boolean Satisfiability Models of Logic Circuits
Srimat Chakradhar, Vishwani Agrawal, and Michael Bushnell
These two recently proposed models of digital circuits are useful in parallel test-generation methods. Moreover, the authors show how the equivalence of these models offers the flexibility of using the same algorithm in different environments.

For subscription information, circle number 200 on the reader service card.

Magazine order form

Please print

Name _____

Address _____

City _____

State/ZIP/Country _____

IEEE-CS member no. (required for discount) _____

Single issue prices:		Quantity	Price	Total
Member, $10	October '90 *Micro*			
Nonmember, $20	October '90 *Expert*			
	October '90 *D&T*			

Mail to:
IEEE Computer Society Order Dept.
10662 Los Vaqueros Circle
PO Box 3014
Los Alamitos, CA 90720-1264

Total
(payment enclosed)

USE THE READER SERVICE CARD TO OBTAIN INFORMATION ON:

- Membership application, student #203, others #202
- Periodicals subscription form for individuals #200
- Periodicals subscription form for organizations #199
- Publications catalog #201
- Compmail II electronic mail brochure #194
- Technical committee list/application #197
- Chapters lists, start-up procedures #193
- Student scholarship information #192
- Volunteer leaders/staff directory #196
- IEEE senior member grade application #204 (requires ten years practice and significant performance in five of those ten)

To check membership status or report a change of address, call the IEEE toll-free number, 1-800-678-4333. Direct all other Computer Society related questions to the Publications Office.

PURPOSE

The IEEE Computer Society advances the theory and practice of computer science and engineering, promotes the exchange of technical information among 100,000 members worldwide, and provides a wide range of services to members and nonmembers.

MEMBERSHIP

Members receive the acclaimed monthly magazine *Computer*, discounts, and opportunities to serve (all activities are led by volunteer members). Membership is open to all IEEE members, affiliate society members, and others interested in the computer field.

IEEE COMPUTER SOCIETY

A member society of the
Institute of Electrical and Electronics Engineers, Inc

PUBLICATIONS AND ACTIVITIES

Computer. An authoritative, easy-to-read magazine containing tutorial and in-depth articles on topics across the computer field, plus news, conferences, calendar, interviews, and product reviews.

Periodicals. The society publishes six magazines and five research transactions. Refer to membership application or request information as noted above.

Conference Proceedings, Tutorial Texts, Standard Documents. The Computer Society Press publishes more than 100 titles every year.

Standards Working Groups. Over 100 of these groups produce IEEE standards used throughout the industrial world.

Technical Committees. More than 30 TCs publish newsletters, provide interaction with peers in specialty areas, and directly influence standards, conferences, and education.

Conferences/Education. The society holds about 100 conferences each year and sponsors many educational activities, including computing science accreditation.

Chapters. Regular and student chapters worldwide provide the opportunity to interact with colleagues, hear technical experts, and serve the local professional community.

OMBUDSMAN

Members experiencing problems — magazine delivery, membership status, or unresolved complaints — may write to the ombudsman at the Publications Office.

EXECUTIVE COMMITTEE

President: Helen M. Wood*
National Oceanic and Atmospheric Administration
FB 4, Rm. 1069, Code E/SP
Washington, DC 20233
(301) 763-1564

President-Elect: Duncan H. Lawrie*
Past President: Kenneth R. Anderson*

VP, Conferences and Tutorials: Laurel V. Kaleda (1st VP)*
VP, Standards: Paul L. Borrill (2nd VP)*
VP, Area Activities: Gerald L. Engel†
VP, Education: Ronald G. Hoelzeman†
VP, Membership and Information: Barry W. Johnson†
VP, Press Activities: James H. Aylor†
VP, Publications: Sallie V. Sheppard*
VP, Technical Activities: Mario R. Barbacci*

Secretary: David Pessel*
Treasurer: Joseph Boykin†
Division V Director: Edward A. Parrish, Jr.†
Division VIII Director: J. T. Cain†
Executive Director: T. Michael Elliott†
*voting member of the Board of Governors
†nonvoting member of the Board of Governors

BOARD OF GOVERNORS

Term Expiring 1990:
Vishwani Agrawal, Mario R. Barbacci
Ming T. (Mike) Liu, Yale N. Patt, Donald E. Thomas,
Benjamin W. Wah, Ronald Waxman

Term Expiring 1991:
P. Bruce Berra, Michael Evangelist,
Ted Lewis, Raymond E. Miller, Earl E. Swartzlander, Jr.,
Joseph E. Urban, Thomas W. Williams

Term Expiring 1992:
Alicja I. Ellis, Tadao Ichikawa,
David Pessel, Sallie V. Sheppard, Bruce D. Shriver,
Harold Stone, Wing N. Toy

Next Board Meeting
March 1, 1991, 8:30 a.m.
Cathedral Hill Hotel, San Francisco, CA

SENIOR STAFF

Executive Director: T. Michael Elliott
Publisher: H. True Seaborn
Director, Conferences and Tutorials: Anne Marie Kelly
Director, Finance and Administration: Tod S. Hesler
Director, Board and Administrative Services: Violet S. Doan

COMPUTER SOCIETY OFFICES

Headquarters Office
1730 Massachusetts Ave. NW
Washington, DC 20036-1903
Phone: (202) 371-0101
Fax. (202) 728-9614

Publications Office
10662 Los Vaqueros Cir
PO Box 3014
Los Alamitos, CA 90720-1264
Membership and General Information:
(714) 821-8380
Publication Orders: (800) 272-6657
Fax: (714) 821-4010

European Office
13, Ave. de L'Aquilon
B-1200 Brussels, Belgium
Phone: 32 (2) 770-21-98
Fax. 32 (2) 770-85-05

Asian Office
Oshima Building
2-19-1 Minami-Aoyama, Minato-ku
Tokyo 107, Japan
Phone. 81 (3) 408-3118
Fax: 81 (3) 408-3553

IEEE OFFICERS

President: Carleton A. Bayless
President -Elect: Eric E. Sumner
Executive Vice President: Martha Sloan
Secretary: Fumio Harashima
Treasurer: Wallace S. Read

VP, Educational Activities: Richard S. Nichols
VP, Professional Activities: Michael J. Whitelaw
VP, Publication Activities: Ralph W. Wyndrum, Jr.
VP, Regional Activities: Robert T. H. Alden
VP, Technical Activities: H. Troy Nagle, Jr.

CALL FOR PAPERS

16th Annual Conference on Local Computer Networks

"The Conference on Practical Leading Edge Computer Networking"

October 14-17, 1991, Minneapolis, Minnesota USA

Sponsored by: IEEE Computer Society TC-Computer Communications

Theme:

The emphasis of this year's conference is on practical experience using local computer networks. This unique approach stimulates a workshop environment and allows for an effective interchange among users, researchers, and vendors. Among the primary goals of the conference are to enable those involved in the local computer network field to share experiences, lessons learned, and prototype data and analysis. Because of these objectives, papers based on experience are especially solicited.

The focus of the 16th LCN Conference will be on **LAN/MAN/WAN Integration** and papers that cover this area are explicitly sought.

Information for Authors:

All authors must submit 5 full copies of the full technical paper. The first page must contain: title of the paper, authors names including affiliations, complete mailing address, telephone and FAX numbers, and a 250 word (max) abstract (double-spaced) in English to Jim Mollenauer, Program Chair at the address below:

Send Papers To:

James F. Mollenauer, Program Chair
16th LCN Conference
Artel Communications
22 Kane Industrial Drive
Hudson, MA 01749
USA

(508) 562-2100-Office
(508) 562-6942-FAX

Sessions are Also Being Organized on:

- *High Speed Networks*
- *FDDI*
- *Integrated Services Networks*
- *LAN Performance*
- *High-Performance Protocols*
- *Metropolitan Area Networks*
- *Internetworking*
- *Standards*
- *Network Management*
- *Real-Time Networks*
- *Gigabit Networks*
- *Routing*
- *Bridges/Routers*
- *ATM*

General Co-Chairs:

Larry Green, Protocol Engines Inc.
Marc Cohn, Raychem Corp.

Program Chair:

Jim Mollenauer, Artel Communications

Important Dates:

Submission: April 5, 1991
Acceptance: June 14, 1991
Camera Copy: August 2, 1991

THE IEEE COMPUTER SOCIETY

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS

For the latest UNIX research & development topics . . .

Winter 1991 USENIX Technical Conference

Grand Kempinski Hotel

Dallas, Texas

January 21-25, 1991

The Professional and Technical UNIX Association

USENIX

For complete conference details, call:
(714) 588-8649 or write:
USENIX Conference Office
22672 Lambert St., Suite 613
El Toro, CA 92630

TUTORIALS

Two days of in-depth tutorials will be offered by internationally renowned UNIX developers and educators. Software professionals and technical managers who want to broaden their expertise in advanced computing systems should register early!

- System Security
- UNIX Systems Internals
- Introductory and Advanced Courses on C++
- Gnu C Compiler Internals
- Graphics and X Windows
- Parallel Programming
- UNIX Technologies in Japan
- Systems Administration & Perl
- Mach
- TCP/IP
- Network Programming & Security

TECHNICAL SESSIONS

Presentations by leading experts on the evolution and development of the UNIX operating system will feature:

- Operating Systems of the Future
- Communications and Networking
- Applications
- User Interfaces
- Programming Environments and Languages
- Testing and Debugging

Special interest sessions offer the attendees an opportunity to exchange information and ideas on such topics as systems administration, security and networking. Facilities featuring Internet and dialout access for attendees is provided.

THE SPONSOR

The USENIX Association is a not-for-profit professional organization dedicated to fostering the development of research and technological information pertaining to UNIX and advanced computing systems.

The 1991 UniForum Trade Show will be held at the Dallas Infomart on the same dates as the USENIX Conference.

UNIX is a registered trademark of AT&T.